PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

ESCUELA DE SISTEMAS

DISERTACIÓN DE GRADO PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS Y COMPUTACIÓN

DESARROLLO DE UN SISTEMA QUE PERMITA LA BÚSQUEDA DE OBJETOS PERDIDOS Y EL REGISTRO DE OBJETOS ENCONTRADOS

AUTOR:

GABRIEL ALMEIDA

DIRECTOR:

ING. DAMIÁN NICOLALDE

Quito, 29 de junio del 2018

Contenido

Índice de ilustraciones

1. Capítulo 1: Antecedentes.

1.1. Justificación

Este proyecto tiene como finalidad el facilitar la interacción entre personas: aquellas que perdieron algún objeto de su propiedad y quieren recuperarlo, y quienes hayan encontrado un objeto perdido y desean devolverlo a su propietario.

Cada día miles de personas salen de sus casas y visitan universidades, empresas y otros espacios públicos e inevitablemente algunas de estas en algún momento perderán algún objeto personal. El problema que se quiere atacar mediante un servicio centralizado que permita obtener y publicar información acerca de propiedad perdida radica en que la gente que extravió algo y quiere recuperarlo no tiene conocimiento de donde buscar y para quien ha encontrado un objeto perdido y quiere retornarlo no sabrá donde hacer pública esa información, por lo que es poco probable que se logre retornar el objeto a su propietario. Como medida paliativa, el individuo puede remitirse al uso de las redes sociales (Facebook, Twitter, etc.) con la esperanza de recuperar lo perdido; el problema radica en que se puede topar con gran cantidad de páginas y es posible que su pedido no llegue muy lejos. Lo mismo sucede cuando alguien encuentra propiedad perdida. Con este sistema, al igual que con una oficina (física) de objetos perdidos, lo que se busca es que la gente sepa dónde buscar y encontrar propiedad extraviada. Teniendo en cuenta que el rol de la tecnología debe ser el de acercar a la gente, así como también de hacer su vida más sencilla se ha propuesto dar una solución tecnológica para este problema.

1.2. Antecedentes

Existen soluciones web en idioma inglés que podrían aprovecharse para este propósito. Algunos ejemplos de estas herramientas son: RepoApp, ChargerBack, ReclaimHub.

- RepoApp

  "RepoApp permite a los negocios y organizaciones gestionar de mejor manera propiedad perdida y encontrada y reclamos de los clientes, en un solo lugar." (Repoapp, sf1, párr. 2).

  Esta aplicación permite ingresar objetos encontrados fácilmente y monitorear su ubicación, ingresar reclamos por objetos perdidos, emparejar los objetos encontrados con los perdidos, tomar fotografías desde un smartphone. Además, ofrece un servicio de reportes. (Repoapp.com, sf2).

- ChargerBack

  Se trata de otra solución web para propiedad perdida y encontrada. Funciona de la siguiente manera: Los clientes reportan su posesión perdida y los negocios registran los objetos encontrados, se emparejan los artículos y se retornan a su dueño mediante el servicio de correo. (ChargerBack, sf).

- ReclaimHub

  "Toma el control de tu propiedad perdida. Maneja los objetos perdidos y encontrados en tu compañía en línea, con nuestro software en la nube". (ReclaimHub, sf1, párr. 1).

  Este sistema se encarga de registrar y rastrear objetos, emparejar artículos perdidos y encontrados y, luego de un tiempo escogido por el usuario, se tiene la opción de donar, reciclar o desechar el artículo hallado. (ReclaimHub, sf2).

En Ecuador no existen soluciones dedicadas al manejo de propiedad perdida, sin embargo, se puede encontrar información en sitios web de avisos clasificados. Como ejemplo tenemos:

- Evisos

  Es una solución web para avisos clasificados. Posee varias categorías de búsqueda, entre ellas tenemos: compraventa, trabajo y empleo, eventos, personas perdidas, objetos perdidos, etc. (Evisos, sf).

- El Universo

  El sitio web del diario El Universo posee un servicio de avisos clasificados. Entre estos se puede encontrar información acerca de objetos perdidos. (El Universo, sf).

Después de ver las soluciones antes mencionadas se ha decidido proceder con el desarrollo de una propia. Una de las razones detrás del impulso de hacer una aplicación original y no usar una ya existente es que se maneja la posibilidad de aprovechar la brecha de oportunidad en el mercado y vender este software a alguna empresa o institución. Aquí entra en juego la formación empresarial de la PUCE en la que se busca emprender en lugar de ser un empleado más de una empresa. Otro motivo para desarrollar una solución nueva es poner en práctica los conocimientos adquiridos durante el período universitario.

La ventaja de este nuevo aplicativo es que está hecho con las últimas herramientas tecnológicas para el desarrollo de aplicaciones web, esto permite a su vez una mayor escalabilidad, puesta punto, manejo de diseño personalizado y seguridades de última generación.

1.3. Objetivos

1.3.1. Objetivo general.

Desarrollar una solución web que permita el manejo de objetos perdidos y encontrados, mediante el uso de una aplicación distribuida.

1.3.2. Objetivos específicos

1.3.2.1. Analizar los requerimientos necesarios, mediante la comprensión del dominio del problema, para la construcción del sistema.

1.3.2.2. Diseñar la arquitectura y los componentes de la aplicación, de modo que funcionen correctamente.

1.3.2.3. Construir una aplicación web amigable con el usuario final, entendiendo las necesidades de este, para que su experiencia sea agradable.

1.3.2.4. Implementar la aplicación haciendo uso de herramientas de desarrollo de software.

1.3.2.5. Probar la aplicación para comprobar que los requerimientos han sido atendidos.

2. Capítulo 2: Marco Teórico.

2.1. Contexto tecnológico.

2.1.1. Sistemas Informáticos

Un sistema de información es un conjunto de elementos que se relacionan entre sí y trabajan colectivamente para procesar datos e información. Estos elementos son: personas que hagan uso del sistema, actividades y recursos materiales. Estos sistemas tienen cuatro tareas básicas: entrada de datos, almacenamiento, procesamiento y salida de información. (Alegsa, Alegsa.com.ar, 2010).

Un sistema informático es un sistema de información que ha sido informatizado, es decir hace uso de herramientas de software y hardware para automatizar procesos. De modo que un sistema informático es el software, hardware y las personas que interactúan con estos para llegar procesar información y obtener resultados. (Alegsa, Alegsa.com.ar, 2016).

Las partes que componen los sistemas informáticos son:

- Recursos de hardware: Son la parte tangible del sistema. Entre estos están computadoras, impresoras, periféricos, placa base, memoria RAM, enrutadores, módems, etc. (¿Qué es hardware y software?, s.f.)

- Recursos de software: Son la parte intangible del sistema. Es un conjunto de instrucciones que le permiten al usuario interactuar con el hardware para realizar tareas. El software se divide en varios tipos: software del sistema (Sistemas operativos, drivers o controladores), software de programación (entornos de desarrollo, compiladores) y software de aplicación (aplicaciones ofimáticas, videojuegos). (Roque, 2014)

- Recurso humano: Son las personas que interactúan con el sistema: programadores, operadores, usuarios finales, stakeholders (involucrados), técnicos. (Alegsa, Alegsa.com.ar, 2016)

Ilustración 1 Recursos de un sistema informático. Definición de Sistema Informático (SI). Recuperado de http://www.alegsa.com.ar

El ciclo de vida típico de un sistema informático consta de los siguientes pasos:

1. **Estudio de factibilidad.** Aquí se analizan las posibilidades de éxito o fracaso del proyecto.
2. **Análisis de requerimientos.** En esta fase se definen los requisitos del sistema.
3. **Diseño.** Es la definición de la arquitectura del sistema.
4. **Implementación.** Aquí se traduce el diseño a una forma que la máquina pueda entender (codificación).
5. **Validación y pruebas.** Consiste en evaluar si el programa cumple las tareas indicadas en la especificación así como de corregir los errores que vayan surgiendo.

6. **Operación y mantenimiento.** En esta etapa es donde el usuario hace uso del sistema. Adicionalmente se puede modificar el producto para corregir errores, aumentar funcionalidades, etc. (Torres, 2014)

El principal objetivo de informatizar un sistema es aumentar la rapidez con la que las tareas son ejecutadas, disminuir errores y reducir costos, tiempo y esfuerzo. (Alegsa, Alegsa.com.ar, 2016).

2.1.2. Aplicación distribuida

Una aplicación distribuida es un software cuyos componentes se hallan en diferentes computadores conectados a una red y que corre como un solo sistema. Los componentes individuales o nodos son considerados como entidades autónomas, cada uno con su propia memoria local, sistema operativo y reloj físico; se comunican entre sí para coordinar sus acciones haciendo uso de protocolos de comunicaciones. La característica definitoria de las aplicaciones distribuidas es que el usuario percibe al sistema como una unidad. (IBM, s.f.1)

Estas aplicaciones presentan ventajas sobre sus contrapartes centralizadas como escalabilidad y redundancia. La escalabilidad hace referencia a que el sistema puede expandirse añadiendo más elementos y la redundancia implica que el trabajo no se detiene si uno de los componentes falla, ya que hay muchos otros que pueden proveer el mismo servicio. (IBM, s.f.2)

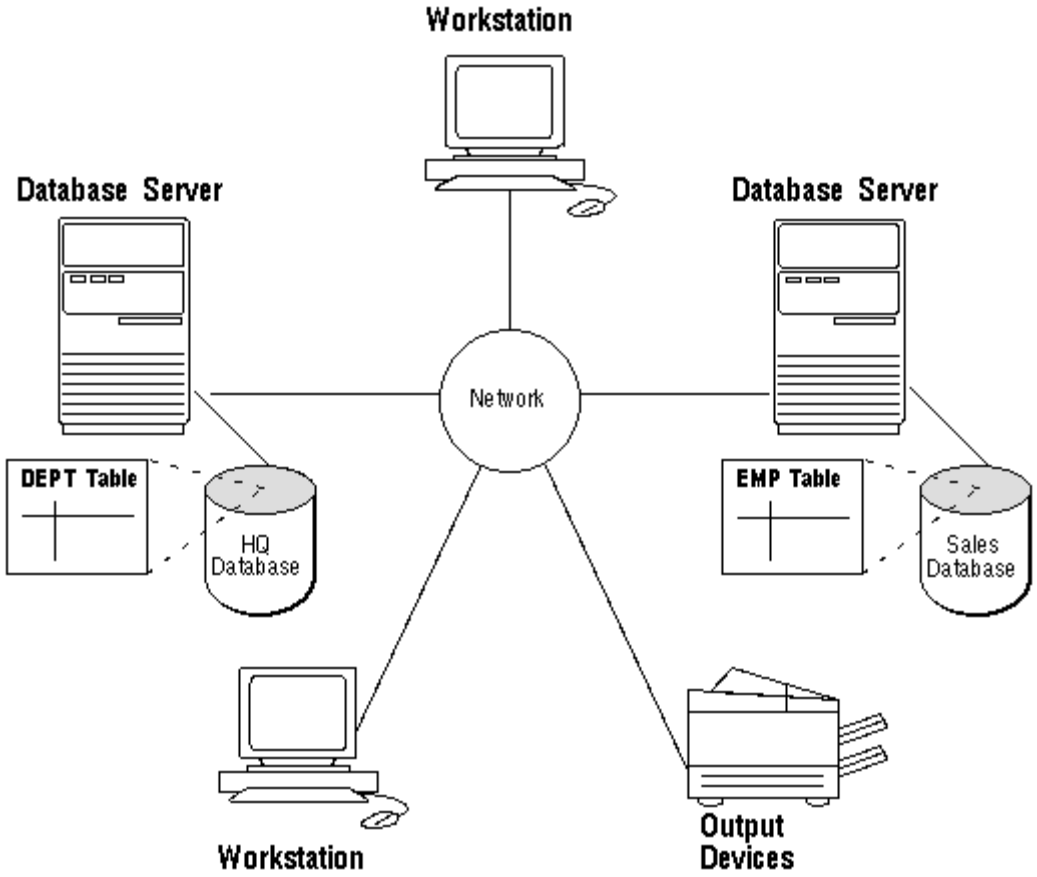

Ilustración 2 Un entorno distribuido. Understanding Distributed Systems. Recuperado de https://docs.oracle.com

Arquitecturas

Las arquitecturas que se usan en aplicaciones distribuidas pueden ser:

- Peer to peer (P2P).

  En esta arquitectura destinada a la comunicación entre aplicaciones, donde la carga de trabajo es dividida entre los componentes llamados pares (peers). Aquí no hay una jerarquía entre nodos, todos son equivalentes en cuanto a permisos, privilegios y en participación en la aplicación. Este tipo de tecnología se usa principalmente para el intercambio de archivos entre usuarios. (Pérez, 2015).

- Cliente – Servidor.

En este tipo de arquitectura existen dos elementos además de la red: el cliente y el servidor. El cliente es el programa con el que interactúa el usuario final a través de los dispositivos de entrada y salida (mouse, teclado) y que envía peticiones al servidor, recibe los resultados y los muestra en pantalla. El servidor está encargado de recibir las peticiones del cliente, procesarlas y servirle los resultados. El cliente y el servidor pueden estar en máquinas separadas, comunicándose a través de una red o pueden residir en el mismo sistema. (Oracle, s.f.1)

En los sistemas cliente – servidor se puede mover procesos e información no críticos al lado del cliente, dejando al servidor con mayor capacidad para procesar las tareas más importantes. Como solamente el servidor tiene acceso a los datos las aplicaciones cliente son independientes de la ubicación física de los datos, si estos son trasladados a otro lugar geográfico las aplicaciones pueden seguir funcionando sin que el cliente se dé cuenta de que hubo algún cambio. De igual manera esto facilita el acceso concurrente a los datos, ya que estos están centralizados en el servidor y no distribuidos en las máquinas cliente. Al utilizar este tipo de sistemas se tiene gran independencia de componentes y se reducen los costos de mantenimiento. (Oracle, s.f.2)

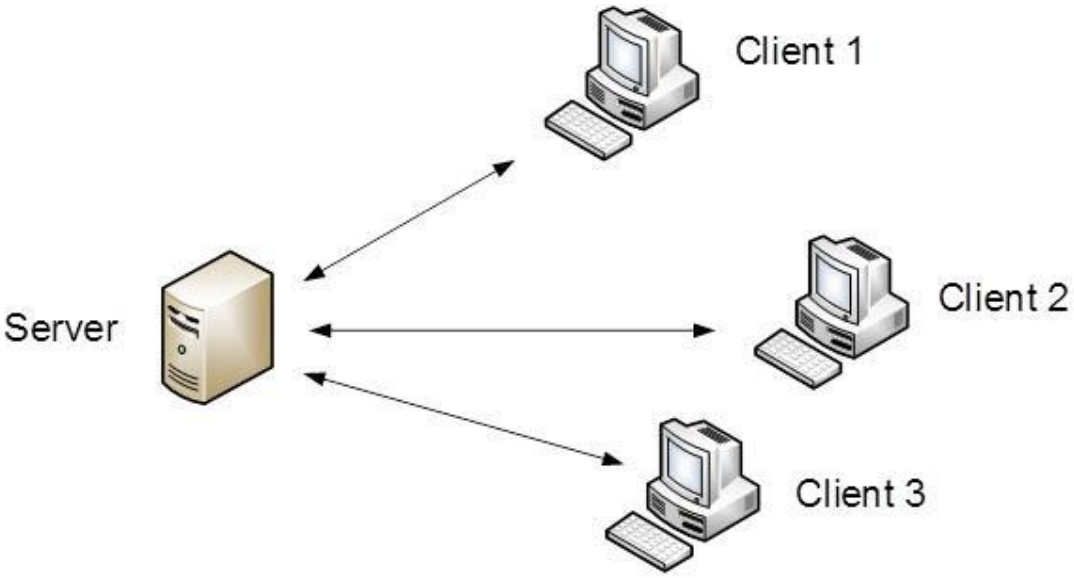

Ilustración 3 Client-Server Topology. Client-Server. Recuperado de https://www.howtonetwork.org

- Tres capas.

Es un tipo de arquitectura cliente – servidor en la que la capa del servidor se subdivide a su vez en dos capas: la capa de aplicación y la capa de datos, que sumadas a la capa de presentación (cliente) constituyen las tres capas. La interfaz de presentación contiene la interfaz gráfica de usuario (GUI) y se comunica con la capa de aplicación. En esta segunda capa se encuentra la lógica del negocio, responsable de las funcionalidades de la aplicación y se comunica con la capa de datos, que provee acceso a la información almacenada. (IBM, s.f.1)

Al momento de la comunicación entre los tres niveles cada uno de estos solo se comunica con la capa adyacente, es decir, la capa de presentación se comunica únicamente con la capa de aplicación y solo ésta puede acceder a la capa de persistencia o de datos. Esta independencia hace más fácil actualizar o modificar a cualquiera de las capas sin impactar a las demás y permite a los desarrolladores trabajar en su área de preferencia en lugar de tratar con todo el sistema. (IBM, s.f.2)

Debido a que es una estructura más compleja puede resultar más difícil de construir y de mantener. Además, la separación física entre capas puede generar problemas al momento de integrarlas. (IBM, s.f.3)

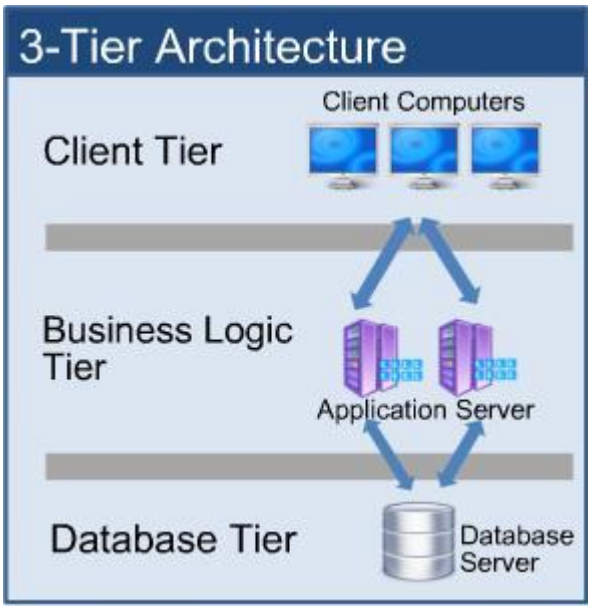

Ilustración 4 Client-Server Topology. Client-Server. Recuperado de https://www.howtonetwork.org

Protocolos de comunicación

Para que una aplicación distribuida funcione sus componentes deben poder comunicarse entre sí satisfactoriamente, esto se da gracias a una comunicación previa, regida por lo que se conoce como protocolos de comunicación, que son un conjunto de reglas o normas que deben cumplir todos los elementos que intervienen en la comunicación a través de la red. Estos protocolos dan un conjunto de descripciones formales de los formatos y reglas a los que están sujetos los mensajes digitales. Estas reglas cubren la sintaxis, semántica, sincronización de la

comunicación, autenticación, detección y corrección de errores, etc. Los protocolos de comunicación son implementados en software y hardware. (Suyama, 2004)

Entre las propiedades de los mensajes digitales que son definidas por los protocolos están: tamaño de paquete, velocidad de la transmisión, mapeo de direcciones, control de flujo, corrección de errores, técnicas de sincronización, ruteo, etc. (Suyama, 2004)

Algunos de los protocolos más conocidos son: User Daragram Protocol (UDP), Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Telnet, Address Resolution Protocol (ARP), entre otros. (Suyama, 2004)

2.1.3. Aplicación web

Es una aplicación distribuida de tipo cliente-servidor en la que el cliente corre en un navegador web que se comunica con el servidor web a través de internet o de una intranet haciendo uso del protocolo HTTP. Sigue una arquitectura de tres capas: la del navegador, la del servidor web y la capa de persistencia, también conocida como capa de datos. El navegador web es el encargado de presentar los recursos al usuario final y de enviar las peticiones realizadas por este al servidor web, que es quien aloja uno o más sitios web en internet. El servidor recibe las peticiones, accede y opera con la capa de datos y envía la página al navegador para su renderización y presentación. (Mozilla, 2018)

Una aplicación web es un sitio web con páginas con contenido sin determinar. Un sitio web es un conjunto de páginas web relacionadas y sus archivos de soporte pertenecientes a un mismo dominio de internet; estas páginas web pueden ser estáticas o dinámicas. Las páginas estáticas presentan únicamente contenido estático HTML, es decir, el diseñador determina el contenido de la página y no cambia cuando esta se solicita al servidor. Las páginas dinámicas en cambio cuando

llega una petición el servidor web en lugar de enviar la página directamente al navegador como lo hace con las páginas estáticas primero lo transfiere a un software llamado servidor de aplicaciones, que es el encargado de finalizar la página. Luego el servidor de aplicaciones accede a la base de datos mediante un controlador o driver, que es quien ejecuta la consulta en la base y devuelve los resultados al servidor, que los inserta en la página y la envía al navegador. (Adobe, s.f.1)

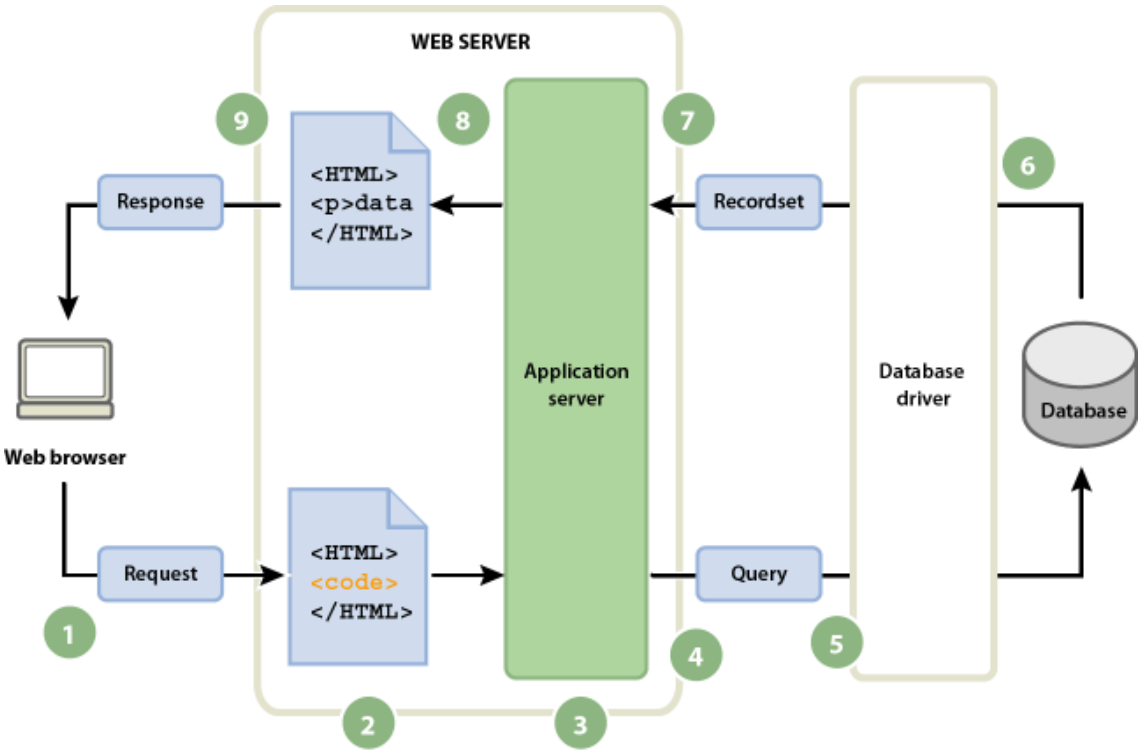

Ilustración 5 Acceso a la base de datos. Web applications. Recuperado de https://helpx.adobe.com

Capa del navegador

Es la capa con la que interactúa el usuario. Hace uso de un software llamado navegador web, que es el que le permite al usuario localizar, acceder y presentar páginas web en el internet. El navegador cumple tres funciones: recuperar, mostrar y navegar. Recuperar o extraer hace referencia a la capacidad del usuario de acceder a los recursos de información provistos por el servidor. Una vez que el navegador recibe los recursos web del servidor renderiza la

información para que pueda ser mostrada al usuario. Finalmente, el navegador permite al usuario navegar por la red (de ahí su nombre), accediendo a otras páginas o sitios web por medio de hipervínculos. (Sass, s.f.1)

Los recursos de información están identificados por un URI (Universal Resource Identifier*)* o identificador universal de recursos. Para acceder a estos, el usuario debe acceder al sitio web ingresando una URL (Universal Resource Locator) en el navegador. El prefijo de la URL indica al navegador como interpretarlo, el más común es http, con lo que los recursos se extraen utilizando el protocolo HTTP (Hypertext Transfer Protocol) o protocolo de transferencia de hyper-texto. (Sass, s.f.2)

HTTP es un protocolo diseñado para la trasmisión de documentos HTML (Hypertext Markup Language) entre el servidor y el navegador web. Es un protocolo sin estado, es decir, no guarda datos entre peticiones. Es sencillo, ya que fue diseñado para ser comprendido fácilmente por los seres humanos, lo que facilita entre otras cosas la depuración de errores. (Mozilla, 2018)

Los navegadores más populares en la actualidad son: Chrome, Safari, Firefox, Internet Explorer – Edge y Opera. En febrero del 2018 Google Chrome domina el mercado de navegadores con un 59,9% de uso. (Browser & Platform Market Share March 2018, 2018)

Algunos de los beneficios de los navegadores web: descargarlos es gratis, se puede tener más de uno instalado en una misma máquina y todos tienen un funcionamiento similar lo que le facilita su uso al usuario final. (Valdes, 2016)

Capa del servidor

Se conoce como servidor web tanto al software que se encarga de servir páginas web al usuario como al hardware en que está alojado dicho software. En cuanto a hardware contiene los archivos que componen el sitio web, entre estos se encuentran: documentos HTML, imágenes, hojas de estilo CSS, etc. La parte del software tiene el control sobre como acceden a los archivos los usuarios. (Mozilla, 2018)

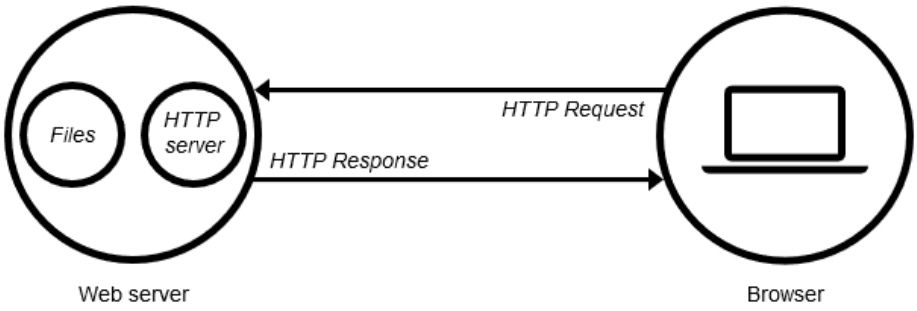

Ilustración 6 Qué es un servidor web? Recuperado de https://developer.mozilla.org

Los servidores web pueden ser estáticos o dinámicos. Los estáticos envían contenido estático HTML y contienen un servidor HTTP y los archivos que componen el sitio web. El servidor HTTP se encarga de procesar y responder las peticiones que vienen del navegador. Siempre debe enviar una respuesta para cada petición, ya sea una página o un mensaje de error, comúnmente el error "404 Not Found". El servidor dinámico añade a esto un servidor de aplicaciones y una base de datos. (Mozilla, 2018)

Capa de datos

Es la capa que provee al servidor acceso a la información almacenada en la base de datos. Consta de un manejador o *driver* y una base de datos. El driver actúa como intermediario entre

el servidor de aplicaciones y la base de datos. Para manejar los datos tiene uno o más sistemas gestores de bases de datos (SGBD). (Mozilla, 2018)

2.2. Técnica de desarrollo de Software.

2.2.1. SCRUM

Es un marco de trabajo ágil que hace énfasis en estrategias de desarrollo incremental, en contraposición a la forma tradicional de planificación y ejecución completa del producto de software. Busca confiar más en la capacidad de las personas de desenvolverse en equipos auto organizados que en la calidad de los procesos empleados. Un principio clave de scrum es que el cliente puede cambiar de opinión sobre lo que quiere en la marcha, lo que se conoce como volatilidad de requerimientos, por lo que la adaptabilidad es clave para los miembros de equipo. (Bailón, 2008)

Scrum define un conjunto de prácticas y roles. Los roles principales son el Product Owner, que es responsable de lo que construye el equipo, el Scrum Master, que se encarga de asegurarse de que el equipo siga el proceso de scrum, así como de resolver impedimentos que surgen en el sprint, y el equipo de desarrollo, que está conformado por los individuos que construyen el producto. (Roche, 2017)

Este tipo de metodología utiliza lo que se conoce como *sprint*, o iteración, que es la unidad base de desarrollo. Cada sprint típicamente dura entre una semana y un mes, y al principio de este el equipo ser reúne para planificar el trabajo que se hará durante ese ciclo. Al final de sprint el equipo revisa las tareas que se completaron y las que fueron planificadas, pero quedaron incompletas. Se

presentan los resultados a los involucrados (solo se muestran las tareas completas), quienes junto con el equipo definen que se va a trabajar en el siguiente ciclo. (Bailón, 2008)

Scrum descansa sobre tres pilares: transparencia, inspección y adaptación. Estos pilares se mantienen gracias a los cinco valores de Scrum (Doshi, 2016):

1. Compromiso – de los miembros del equipo de alcanzar las metas de cada sprint.
2. Coraje - para perseverar a través de las dificultades y los retos que ofrece el camino.
3. Atención - los miembros deben concentrarse en las tareas definidas para la iteración actual.
4. Franqueza - debe haber sinceridad entre el equipo y los involucrados.
5. Respeto – por las aptitudes y diferencias entre miembros. (Spear, 2018)

En cuanto a limitaciones, Scrum no funciona de manera tan eficiente en las siguientes circunstancias:

- Cuando hay poco contacto entre los miembros del equipo. Según el manifiesto ágil, la mejor manera de comunicarse es cara a cara.
- Cuando los integrantes tienen habilidades muy específicas. Muchas veces es necesario que un integrante continúe el trabajo de otro miembro del equipo. (Doshi, 2016)

2.3. Herramientas de desarrollo

2.3.1. Xampp

En un conjunto (*stack*) de soluciones del lado del servidor, libre y de código abierto, desarrollado por Apache Friends. Está conformado por el servidor HTTP Apache, el sistema gestor de bases de datos MySQL y los lenguajes de programación PHP y Perl. El nombre Xampp es un acrónimo, donde X representa multi-plataforma, la A de Apache, M de MySQL y las dos P de PHP y Perl

respectivamente. Además, incluye un servidor SMTP (para manejo de correos electrónicos) y un FTP (para transferencia de archivos). (Thakral, 2011)

El principal beneficio de esta tecnología es su facilidad de instalación, con lo que no es necesario instalar sus componentes de manera separada, lo que facilita mucho la creación de un servidor web local para desarrollar y hacer pruebas. (Thakral, 2011)

2.3.2. Apache

Es el servidor web más popular en la actualidad. Auspiciado por The Apache Software Foundation, su motivación es proveer un servidor web seguro, extensible y eficiente que provea servicios HTTP. Fue creado en 1995 y para el 2009 servía a más de 100 millones de sitios web. En 2016, se estimó que Apache sirve el 46% de todos los sitios web activos. (Apache, s.f.1)

2.3.3. PHP

PHP: Hyper-Text Preprocessor (antiguamente Personal Home Page) es un lenguaje de script del lado del servidor que se usa en el desarrollo de páginas web dinámicas. Su principal característica es su flexibilidad: corre en varias plataformas (Windows, Linux, Mac OS, etc.), es compatible con la mayoría de los servidores web, como Apache o IIS (Internet Information Services), y soporta un gran rango de bases de datos. Esta flexibilidad tiene sus desventajas, la facilidad de aprendizaje y la cantidad de soluciones posibles para un mismo problema pueden derivar en código espagueti (un montón de hilos anudados e intrincados). (W3Schools, 2017)

Entre las funcionalidades de PHP se encuentran:

- Generar contenido dinámico.
- Administrar archivos en el servidor.
- Manejar información de la base de datos.
- Controlar el acceso de usuarios.
- Encriptar información. (W3Scools, 2017)

2.3.4. MySQL

Es un sistema de gestión de bases de datos de código abierto, basado en el lenguaje estructurado de consultas SQL. Es multiplataforma, se ejecuta en entornos Linux, Windows, UNIX, etc. Se lo asocia principalmente con el desarrollo web, pero puede usarse en una amplia gama de aplicaciones. MySQL fue adquirida por Oracle, mas la licencia GNU se mantiene. A partir de MySQL se desarrollaron sistemas derivados (*forks*): Drizzle, MariaDB y Percona Server con XtraDB. (Rouse, 2015)

2.3.5. Firefox

Es un navegador web libre y de código abierto desarrollado por la Fundación Mozilla. Es multiplataforma y está disponible en Linux, Android, IOS, OS X y Windows. Las características que ofrece este navegador incluyen: navegación por pestañas, corrector ortográfico, restauración de sesión, protección anti-phishing y contra programas espías, actualizaciones automáticas, complementos y plugins, entre otros. (Rouse, 2006)

2.3.6. MVC

El Modelo – Vista – Controlador es un patrón de arquitectura de software que divide las aplicaciones en tres partes interconectadas. Es muy popular en el mundo del desarrollo web. Los tres componentes de la arquitectura MVC son el modelo, la vista y el controlador. El modelo se encarga de la parte de la información, consultando la base de datos. Maneja también la lógica y las reglas de la aplicación. El controlador responde a las acciones del usuario e interactúa con la información del modelo. La vista presenta el modelo en un formato adecuado para su visualización por parte del usuario. (Hernández, 2015)

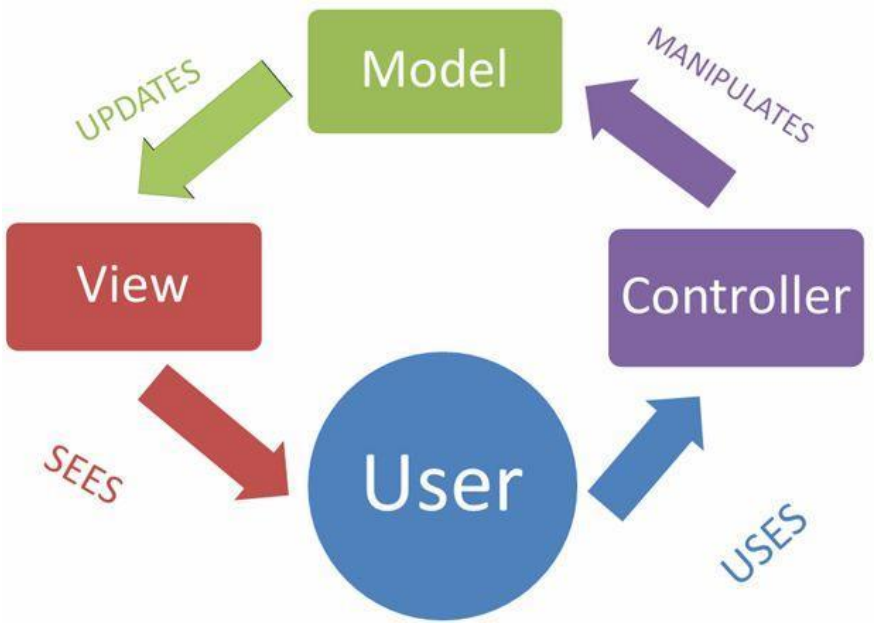

Ilustración 7 Model-View-Controller. Crear Sitio Web. Recuperado de https://nicolasgemio.wordpress.com

2.3.7. Yii2 Framework

Es un framework (marco de trabajo) de PHP de alto rendimiento, basado en componentes y diseñado para construir aplicaciones web modernas en poco tiempo. Implementa el patrón de diseño MVC (modelo – vista – controlador). El nombre Yii significa *simple y evolutivo* en chino, y también puede verse como un acrónimo de *Yes It Is* (Sí, lo es en español). (¿Qué es Yii?, 2014)

Las características de Yii incluyen:

- Facilidad de instalación. Utiliza el manejador de dependencias Composer, que facilita en gran medida la instalación del framework.
- Altamente extensible.
- Enfoque a la seguridad. Yii viene con un componente con varios métodos que ayudan a construir una aplicación más segura.
- Alto rendimiento. (¿Qué es Yii?, 2014)

3. Capítulo 3 – Metodología de desarrollo y Diseño

Para el desarrollo del aplicativo se utilizó la metodología Scrum, y se ha establecido un plazo de dos meses, con fecha de inicio el primero de febrero del 2018 y su finalización 28 de marzo del mismo año. En el proceso se determinaron dos sprints (ciclos), uno para cada funcionalidad del aplicativo: usuarios y objetos. A cada uno de estos ciclos le corresponderá un mes para su culminación. Se ha determinado también una carga de trabajo diaria de 3 horas, con 28 días por cada mes se tienen 84 horas por sprint. Debido a que una misma persona ocupa los varios roles que tiene la metodología Scrum se ha establecido que no existirán reuniones diarias (*meetings*). Las historias de usuario y los ciclos se detallarán a continuación:

3.1. Historias de usuario

Tabla 1: Historias de usuario

Enunciado de la Historia				Criterios de aceptación		
ID de la historia	Rol	Característica/Funcionalidad	Razón/Resultado	Contexto	Evento	Resultado/Comportamiento esperado
1	Como un usuario general	Necesito acceder a un listado de objetos perdidos y en venta	Con la finalidad de visualizar los objetos que han sido reportados o están en venta	Acceso a sitio público con vista de objetos en estado de perdidos o en venta	Ingresar a las ventanas de objetos perdidos o en venta	Despliegue del listado de objetos perdidos y en venta
2	Como un usuario general	Necesito poder ver cada objeto individualmente	Con la finalidad de visualizar los detalles de cada objeto	Acceso a sitio público con vista de detalles del objeto	Ingresar a la ventana de detalle de objeto	Despliegue de los detalles del objeto
3	Como un usuario general	Necesito contactar vía email a quien reportó el objeto	Con la finalidad de reclamar o comprar el objeto	Acceso a sitio público con formulario para envío de correo electrónico	Llenar el formulario de contacto	Envío de email con los datos de quien lo envía al usuario que reportó el objeto
4	Como un usuario general	Necesito crear una cuenta en el sitio.	Con la finalidad de poder administrar mis objetos.	Acceso a sitio con formulario de registro	Llenar el formulario de registro	Creación de una cuenta de usuario con confirmación por email
5	Como un usuario registrado	Necesito reportar un objeto perdido	Con la finalidad de devolver o poner en venta el objeto	Acceso a sitio con formulario de reporte de objeto	Llenar el formulario de reporte	Ingreso al sistema del objeto reportado
6	Como un usuario registrado	Necesito tener un listado de mis objetos de acuerdo a su estado	Con la finalidad de visualizar mis objetos perdidos, devueltos, en venta y vendidos	Acceso a sitio con las listas de objetos ingresados por un usuario	Ingresar a cualquiera de las listas de objetos	Despliegue del listado de objetos de acuerdo a su estado

7	Como un usuario registrado	Necesito poder iniciar sesión en el sistema	Con la finalidad de acceder a mi cuenta de usuario	Acceso a sitio con formulario de inicio de sesión	Llenar el formulario de inicio de sesión	Despliegue de menú de administración de objetos
8	Como un usuario administrador	Necesito administrar los usuarios registrados	Con la finalidad de agregar, editar o eliminar usuarios	Acceso a sitio con listado de usuarios	Ingresar a la lista de usuarios	Despliegue del listado de usuarios con opciones de creación, edición y eliminación

3.2. Sprint 1: Funcionalidad de usuarios

En este primer ciclo se desarrollarán las actividades que conciernen al manejo de usuarios. Este, como los demás ciclos, cuenta con un período de cuatro semanas completas para su realización, iniciando el 1 de febrero y culminando el 28 de febrero. Las actividades de este sprint, un detalle de estas y los tiempos estimados para su realización se detallan a continuación:

Tabla 2: Sprint 1: Funcionalidad de usuarios

ID	Etapa	Tarea	Detalle	Estimación (horas)
S1-1	Análisis	Requerimientos de usuario	Requerimientos de esta funcionalidad	4
			Total	4
S1-2	Diseño	Diseño de arquitectura	Definir la arquitectura apropiada para el aplicativo	2
		Diseño de la base de datos	Definir la entidad	2

			Definir atributos	2
		Definición del modelo		2
		Definición del controlador CRUD		2
		Definición de vistas		3
		Login	Modelos, controladores, vistas	3
		Registro de usuarios	Modelos, controladores, vistas	2
		Perfiles de usuario		2
			Total	20
S1-3	Implementación	Codificación del modelo		6
		Codificación del controlador		7
		Codificación de las vistas		7
		Implementación del login		6
		Implementación del registro de usuarios		8
		Implementación de los perfiles de usuario		6
			Total	40
S1-4	Pruebas	CRUD de usuarios		5
		Login		5
		Registro		5
		Perfiles de usuario		5
			Total	20

Tiempo total estimado para el primer sprint: 84 horas.

3.3. Sprint 2: Funcionalidad de objetos

Este segundo sprint se dará inicio el 1 de marzo y culminará el 28 de dicho mes. Como en el anterior tampoco habrá reuniones diarias. Las actividades que se realizarán en esta etapa tienen que ver con la funcionalidad de objeto. En la siguiente tabla se detallan las actividades y sus tiempos estimados para completar el segundo sprint.

Tabla 3: Sprint 2: Funcionalidad de objetos

ID	Etapa	Tarea	Detalle	Estimación (horas)
S1-1	Análisis	Requerimientos de objetos	Requerimientos de esta funcionalidad	4
			Total	4
S1-2	Diseño	Diseño de la base de datos	Definir la entidad	3
			Definir atributos	3
			Definir la relación	3
		Definición del modelo		2
		Definición del controlador CRUD		2
		Definición de vistas		3
		Notificaciones vía email		2
		Seguridades de acceso		2

			Total	20
S1-3	Implementación	Codificación del modelo		8
		Codificación del controlador		8
		Codificación de las vistas		8
		Implementación de las notificaciones		8
		Implementación de seguridades de acceso		8
			Total	40
S1-4	Pruebas	CRUD de objetos		6
		Notificaciones vía email		7
		Seguridades de acceso		7
			Total	20

Tiempo total estimado para el segundo sprint: 84 horas.

4. Capítulo 4: Implementación

Para el desarrollo del aplicativo se utilizó la metodología Scrum debido a su flexibilidad. El sistema se desarrolló en dos sprints (ciclos), uno para cada funcionalidad, los cuales se detallarán a continuación:

4.1. Sprint 1: Funcionalidad de usuarios.

4.1.1. Análisis:

En esta etapa del primer sprint, se determinaron los requerimientos que debe cumplir la funcionalidad de usuarios:

- El sistema permitirá la administración de usuarios.
- El sistema hará posible el inicio de sesión de usuarios.
- El sistema tendrá un sistema de registro de usuarios.
- El sistema proveerá perfiles a los usuarios.

En base a los requerimientos mencionados anteriormente, se prosiguió a la segunda etapa:

4.1.2. Diseño:

En esta etapa se definieron los componentes de la funcionalidad de usuarios:

4.1.2.1. Diseño arquitectónico

Debido a que se trata de una aplicación web la arquitectura correspondiente al aplicativo es cliente-servidor. Junto con programación en el framework Yii2, que usa el patrón de diseño Modelo-Vista-Controlador (MVC) desarrollado en PHP.

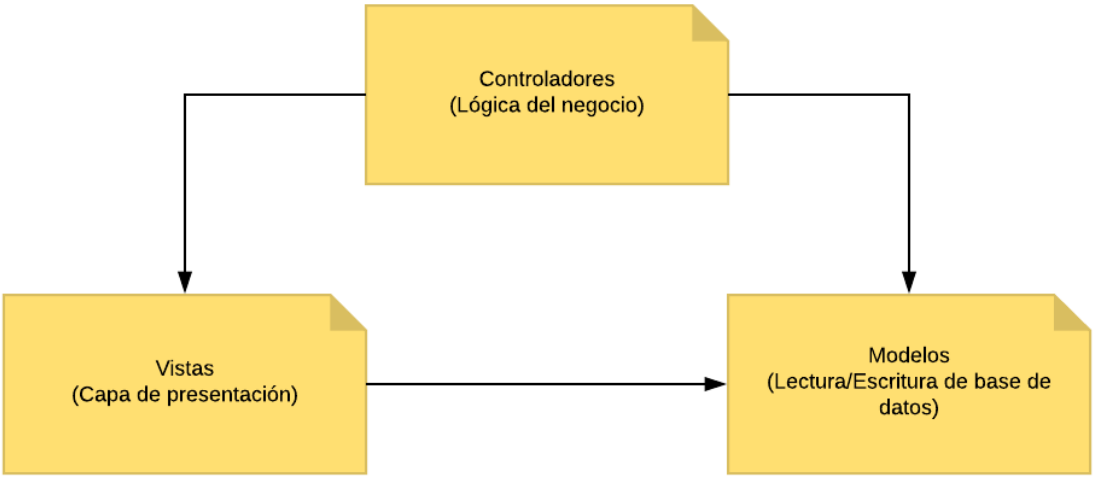

Ilustración 8 Arquitectura de la aplicación.

4.1.2.2. Diseño de la Base de Datos

Se definió la entidad de usuarios llamada user, con sus atributos.

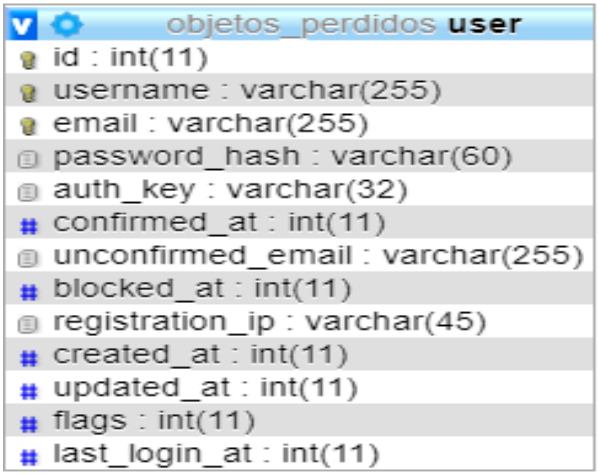

Ilustración 9 Tabla de usuario

4.1.2.3. Definición del modelo

Para la entidad e usuarios se definió una clase básica para la persistencia llamada User, que contiene los atributos que están en la base de datos. Se definió una función para poder crear usuarios con confirmación por email. Además se incluye un método que contiene las etiquetas de los atributos y uno que contiene las reglas que deben cumplir, como longitud, unicidad, obligatoriedad, etc.

4.1.2.4. Definición del controlador

Se definió un controlador de usuarios que contiene las acciones que permiten hacer CRUD de usuarios: créate, index, update y delete.

4.1.2.5. Definición de vistas

Cada acción presente en el controlador tendrá una vista propia: créate, index y update. No es necesaria una vista para la acción delete.

4.1.2.6. Diseño del login

Modelo

En el modelo se incluirán los métodos con las reglas que deben cumplir los atributos: unicidad de correo electrónico y nombre de usuario, etc. Así como también métodos de validación de atributos y para inicio de sesión.

Controlador

Aquí se incluirá las funciones que permitan hacer login y logout.

Vista

Se implementará una vista única para la parte de login.

4.1.2.7. Diseño del Registro de usuarios

Modelo

En el modelo se incluye el método que define las reglas que deben cumplir los atributos y la función que permite hacer el registro de usuarios.

Controlador

En el controlador se incluirá la acción que permite registrar los usuarios.

Vista

De igual manera que en el caso del login se tiene una sola vista para la funcionalidad de registro de usuarios.

4.1.2.8. Diseño de los perfiles de usuario

Se definieron tres perfiles para los usuarios del sistema:

- Administrador: tiene acceso completo al sitio, incluyendo la administración de usuarios registrados.
- Usuario registrado: tiene acceso parcial al sitio, puede administrar objetos en el sistema.
- Usuario público: solo tiene acceso a las vistas de objetos perdidos y en venta.

Se implementarán las acciones en el controlador que permitan hacer el CRUD de perfiles, así como vistas para cada una de estas acciones (excepto delete).

4.1.3. Implementación

En esta etapa consistió en la codificación de los componentes definidos en la etapa de diseño.

4.1.3.1. Implementación del modelo

Create - Esta función es la responsable de la creación de un nuevo registro por medio del uso del modelo de usuario para la inserción en base de datos.

Attribute Labels – En este método se especifican las etiquetas de los atributos de la clase User que se mostrarán en pantalla al usuario para facilitar su comprensión. Por ejemplo el atributo "last_login_at" se muestra en pantalla como "Ultimo login".

Rules – Aquí se dan las reglas que deben cumplir los atributos para nombre de usuario, email y contraseña. Las reglas incluyen obligatoriedad, unicidad, longitud máxima, y eliminado de espacios antes y después (trim).

4.1.3.2. Implementación del controlador

Create – Esta acción hace un llamado al método de creación del modelo definido anteriormente para la inserción de un registro en la tabla de usuarios. Para poder hacerlo renderiza una vista para creación de usuarios.

Index – En esta acción solamente se redirecciona hacia la vista de índice de usuarios, que es renderizada en este método.

Update – Esta acción hace un llamado al modelo de usuario y, luego de hacer la validación de los atributos guarda los cambios y actualiza la página.

Delete – En esta función se elimina el registro del usuario teniendo la precaución de que un usuario no pueda borrarse a sí mismo. Una vez eliminado se redirecciona a la vista de index.

4.1.3.3. Implementación de la vista

Create – Esta es la vista para creación de usuarios. Los atributos mínimos que se deben proporcionar para crear un usuario son el email, el nombre de usuario y la contraseña. Como información adicional se pueden proveer detalles del perfil del usuario como su nombre, teléfono, país de residencia, etc.

Index – En esta vista se puede apreciar los usuarios registrados en el sistema. Se presentan en una tabla con sus atributos principales (nombre de usuario y email). Además se muestran atributos que ayudan al momento de administrar usuarios, como hora de registro, último login, etc.

Update – Aquí se muestra la misma vista que al momento de crear el usuario, pero solo permite modificar los campos de un usuario.

4.1.3.4. Implementación del login

Modelo

Rules – En este método se validan los atributos al momento de hacer login. Además se provee la opción de recordar el usuario actual para futuros inicios de sesión.

ValidatePassword – Hace un llamado a un método provisto por el framework para validación de contraseñas. Se valida tanto el nombre de usuario como la contraseña.

Login – La función responsable de hacer el inicio de sesión propiamente dicho luego de verificar la validación de los datos. Además guarda la fecha en que se inició sesión por última vez.

Controlador

Login – Esta acción llama al método de login definido en el modelo. Se hace una validación Ajax del modelo y se inicia sesión. Renderiza la vista de login.

Logout – En esta función se cierra sesión y se redirecciona al usuario a la pantalla principal de la aplicación.

Vista

Login – En esta vista se tiene un recuadro donde se llena el nombre de usuario y la contraseña. Además de un checkbox para recordar el usuario y el botón de inicio. Adicionalmente se muestran dos enlaces, uno para pedir otro mensaje de confirmación y otro para registrarse en caso de no poseer una cuenta.

4.1.3.5. Implementación del registro de usuario

Modelo

Rules - Aquí se dan las reglas que deben cumplir los atributos para nombre de usuario, email y contraseña. Las reglas incluyen obligatoriedad, unicidad, longitud máxima, y eliminado de espacios antes y después (trim).

Register – Esta es la función que permite hacer el registro de usuarios por medio del modelo a la base de datos y envío de confirmación por correo electrónico.

Controlador

Register – Esta acción renderiza la vista de registro y hace llamado a la función de para registrar definida en el modelo para la creación de una nueva cuenta de usuario.

Vista

Register – En esta vista se muestra un recuadro donde se ingresan el email, el nombre de usuario y la contraseña para hacer el registro, seguido del botón de registrar y un enlace para iniciar sesión en caso de que el usuario esté ya registrado en el sistema.

4.1.3.6. Implementación de perfiles de usuario

Se implementaron los controladores que permiten administrar los roles:

Index – Esta acción accede al modelo y renderiza la vista del índice de roles.

Create – Esta acción accede al modelo y renderiza la vista de creación de roles.

Update – Recibe el nombre del rol a actualizar y guarda los datos modificados. Renderiza la vista de update.

Delete – Acción que recibe como argumento el nombre del rol y lo elimina. Al final redirecciona a la vista del índice de roles.

Así mismo, se implementaron las vistas:

Index – Prersenta una tabla con el nombre y la descripción de los roles.

Create – Esta vista se compone de un formulario para el llenado de los atributos de un nuevo rol.

Update – Esta vista tiene el mismo formulario de la vista anterior donde se pueden modificar los atributos existentes de un rol.

4.1.4. Pruebas

Una vez concluida la implementación, se prosiguió a hacer las pruebas para comprobar el correcto funcionamiento de la funcionalidad. Se hicieron las pruebas de caja blanca, caja negra y la de los criterios de aceptación.

4.1.4.1. Caja blanca

También conocidas como de caja transparente, se hace un análisis del código para ver posibles errores y medir la cobertura del código. Yii2 provee una herramienta para pruebas de este tipo utilizando el framework de pruebas Codeception. Luego de crear una base de datos especialmente para este propósito y de hacer las configuraciones necesarias se

pudieron hacer las pruebas de aceptación, que evalúan escenarios comunes utilizando emulación de un navegador web.

4.1.4.2. Caja negra

En esta prueba se evaluó el sistema usando entradas y salidas. Se ingresaron valores límite en los atributos al momento de administrar usuarios, registrarse y hacer login teniendo resultados satisfactorios en todos los casos.

4.1.4.3. Criterios de aceptación

Para esta parte de las pruebas, el usuario probó los tres criterios de aceptación pertenecientes a la funcionalidad de usuarios:

4.1.4.3.1 Despliegue de la lista de usuarios

Este criterio de aceptación está relacionado con la necesidad de un usuario autorizado de administrar las cuentas de los usuarios registrados, para lo cual es necesario el listado de estos con opciones de creación, edición y eliminación.

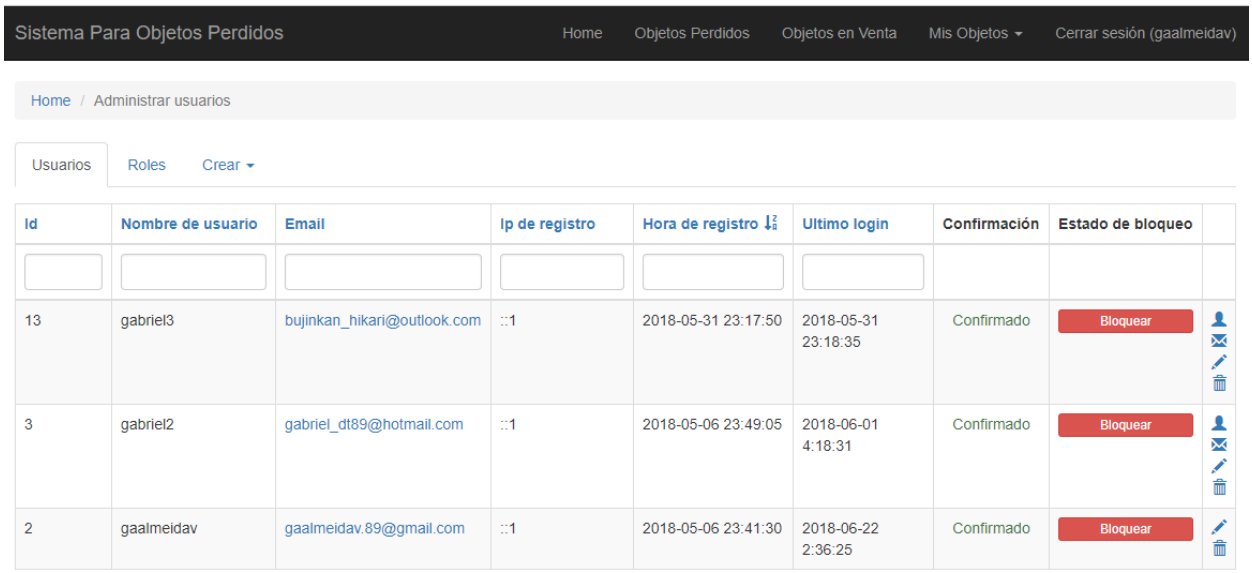

Ilustración 10 Vista del índice de usuarios

Como se puede apreciar en el gráfico, se tiene el listado con los usuarios registrados, y en la parte derecha de la tabla se encuentran los íconos que permiten editar y eliminar usuarios, mientras que la opción de crear un usuario se encuentra en una de las pestañas arriba de la tabla de usuarios.

4.1.4.3.2. Creación de una cuenta de usuario

Para poder administrar objetos, un usuario general debe primero crear una cuenta en el sistema, con esto pasa a ser un usuario registrado. Después de llenar el formulario de registro con los campos de email, nombre de usuario y contraseña se recibió un correo electrónico de

confirmación, después de hacer click en el vínculo ahí presentado se tiene que la cuenta de usuario fue creada y se pudo iniciar sesión con normalidad.

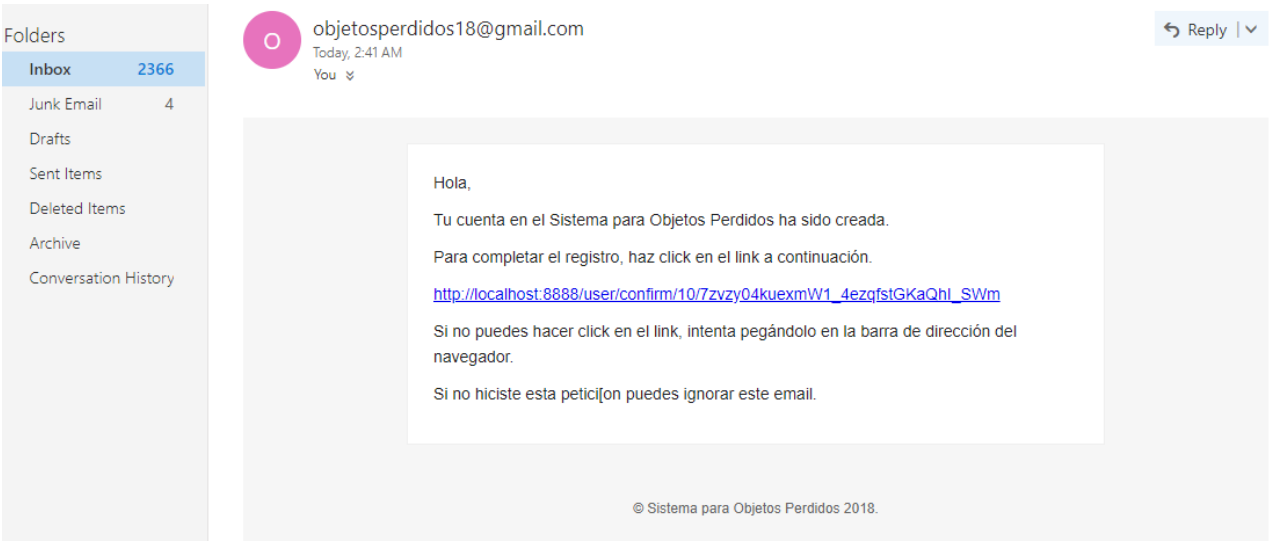

Ilustración 11 Vista de notificación de registro

4.1.4.3.3. Despliegue de menú de administración de objetos

Esta prueba se llevó a cabo para cubrir la necesidad de un usuario registrado de iniciar sesión. Para poder aceptar como válida esta parte de la funcionalidad, se debe tener acceso a un menú desplegable con las opciones de administración de objetos, lo que fue comprobado por el usuario.

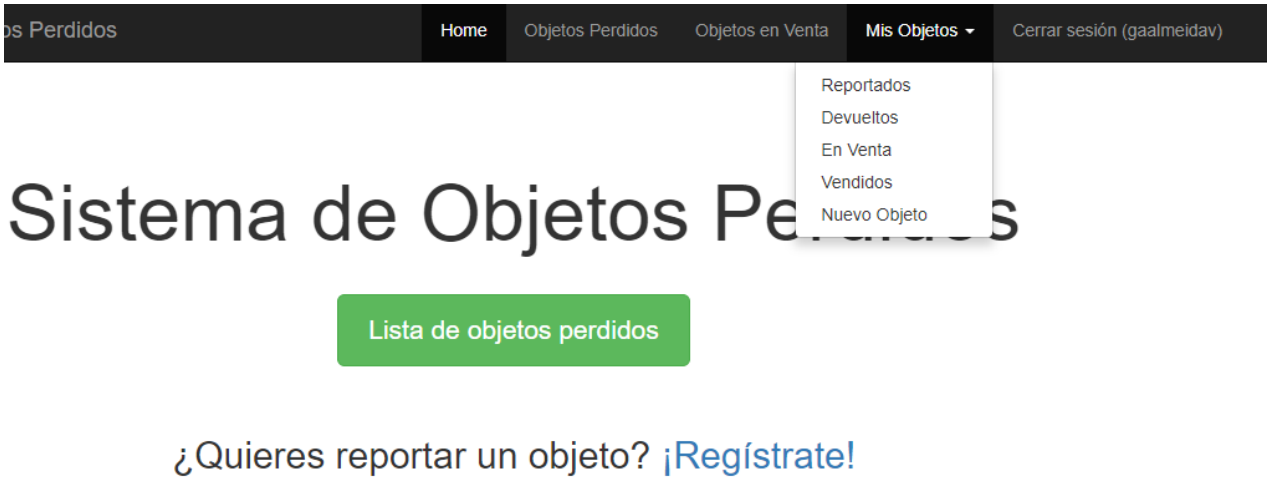

Ilustración 12 Menú de administración de objetos

4.2. Sprint 2: Funcionalidad de objetos.

4.2.1. Análisis

En esta primera etapa del segundo sprint se definieron los requerimientos que debe cumplir la funcionalidad de objetos.

- El sistema permitirá administrar objetos perdidos y encontrados.
- El sistema hará posible el envío de notificaciones vía email.
- El sistema contará con seguridades de acceso.

4.2.2. Diseño

Se procedió al diseño de los componentes de la funcionalidad.

4.2.2.1. Diseño de la base de datos

Además de la tabla para usuarios se definió una para el manejo de objetos con sus atributos respectivos. La relación entre las dos entidades es de uno a muchos, ya que un usuario puede manejar varios objetos. La base de datos queda entonces constituida de la siguiente manera:

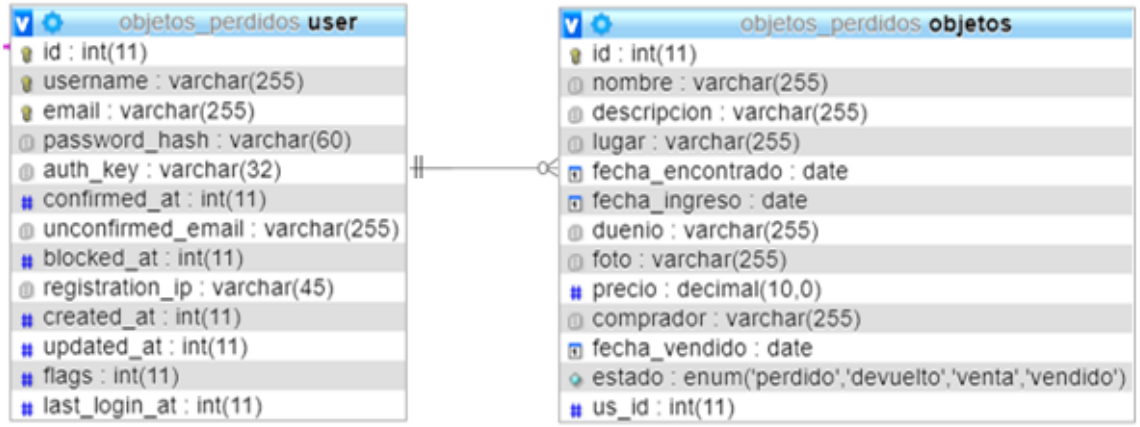

Ilustración 13 Base de datos

4.2.2.2. Definición del modelo

Para la entidad de objetos se definió una clase básica para la persistencia llamada Objetos, que contiene los atributos que están en la base de datos. Se incluye un método que contiene las etiquetas de los atributos y uno que contiene las reglas que deben cumplir, como longitud, unicidad, obligatoriedad, etc. Además se incluye un método search, que permite hacer una búsqueda dinámica de objetos.

4.2.2.3. Definición del controlador

Se definió un controlador de objetos que contiene las siguientes acciones:

General – muestra todos los objetos perdidos o en venta pertenecientes a todos los usuarios.

Index - Muestra todos los objetos pertenecientes al usuario que se encuentra en sesión, dependiendo de su estado.

View – para ver los detalles del objeto.

Update – para actualizar los datos del objeto.

Delete – para borrar un objeto.

Actualizar – esta función permite actualizar el estado del objeto, de perdido a devuelto o a venta y de venta a vendido.

4.2.2.4. Definición de las vistas

Se implementarán vistas para las acciones presentes en el controlador, con excepción de delete y actualizar.

4.2.2.5. Diseño de las notificaciones vía email

Se implementará en el modelo una función que envíe los correos electrónicos, que será llamada por la acción presente en el controlador que permita contactar. Por último se implementará la vista que será mostrada en la bandeja de entrada del usuario. Para poder enviar las notificaciones se creará una cuenta de correo en gmail llamada objetosperdidos18@gmail.com.

4.2.2.6. Diseño de las seguridades de acceso

Para evitar que usuarios no autorizados accedan a información que no les corresponde se definieron tres roles con sus respectivas reglas:

- Administrador: tiene acceso completo al sitio, incluyendo la administración de usuarios registrados.
- Usuario registrado: tiene acceso parcial al sitio, puede administrar objetos en el sistema.
- Usuario público: solo tiene acceso a las vistas de objetos perdidos y en venta.

Las acciones para hacer esto posible serán implementadas en el controlador, con sus respectivas vistas.

4.2.3. Implementación

En la fase de implementación se procedió a codificar los componentes definidos en la etapa de diseño.

4.2.3.1. Implementación del modelo

Rules - Aquí se dan las reglas que deben cumplir los atributos para nombre, descripción, lugar, foto, estado e id de usuario. Las reglas incluyen tipo de dato, obligatoriedad, unicidad y longitud máxima.

Attribute Labels – En este método se especifican las etiquetas de los atributos de la clase Objeto que se mostrarán en pantalla al usuario para facilitar su comprensión.

4.2.3.2. Implementación del controlador

General – Esta acción se encarga de mostrar en pantalla el listado de objetos perdidos o en venta, para lo cual recibe un filtro como argumento y accede al modelo que a su vez hace uso de la base de datos. Renderiza una vista con el mismo nombre.

Index – Este método recibe un filtro como argumento, accede al modelo y muestra en pantalla el listado de objetos de acuerdo a su estado.

View – Acción que recibe la identificación del objeto y muestra sus detalles en una vista que es renderizada en el método.

Create – En este método se instancia un objeto del modelo de usuarios y se crea un nuevo registro en la tabla de objetos con el id del usuario como clave foránea del objeto. Redirecciona a la vista de detalles del objeto.

Update - Esta acción recibe el id del objeto y encuentra el modelo, luego de hacer la validación de los atributos guarda los cambios y redirecciona a la vista de detalle de objeto.

Delete – Acción que recibe el id de un objeto y lo elimina de la base de datos. Redirecciona al índice de objetos.

Actualizar estado – Recibe como argumento el id y el estado del objeto y de acuerdo a ese estado lo actualiza por medio de una estructura de control: de perdido a devuelto o venta y de venta a vendido. Redirecciona a la vista de detalle del objeto.

4.2.3.3. Implementación de las vistas

Create – En esta vista se renderiza un formulario con los atributos que debe tener el objeto a ser creado. Todos son obligatorios.

Update - Aquí se muestra la misma vista que al momento de crear un objeto, pero solo permite modificar los campos del mismo.

Ambas vistas, la de creación y actualización, llaman a un formulario que provee los campos a ser llenados o editados dependiendo de su estado.

General – Esta vista contiene una tabla con el listado de objetos perdidos o en venta de todos los usuarios y sus atributos principales: nombre, descripción, lugar y fecha de ingreso.

Index - Esta vista contiene una tabla con el listado de objetos de acuerdo a su estado de un solo usuario con sus atributos principales: nombre, descripción, lugar y fecha de ingreso.

View – Esta vista contiene una cuadrícula con los detalles de un objeto: nombre, descripción, lugar, fecha y foto. Todos los objetos, independientemente de su estado tienen los botones de actualizar y eliminar. Además de eso, si el objeto se encuentra como perdido aparecen los botones de devolver y poner en venta. Si ha sido puesto en venta aparece el botón de vendido.

4.2.3.4. Implementación de las notificaciones

Configuración del framework

Para poder enviar las notificaciones vía correo electrónico, primero se configuró el framework. En el archivo de configuración se estableció el parámetro ‘useFileTransport’ como *false*, de otro modo el framework almacenaría los correos en un archivo. Además, se estableció el servidor SMTP que se usó y se dieron la dirección del correo desde donde se enviarán los mails y su contraseña.

Codificación de las notificaciones

Luego vino la parte de la codificación. Se comenzó implementando en el modelo la función que envía los correos electrónicos:

Contact – Función que recibe como argumento el email de quien quiere hacer el contacto y el objeto que se quiere reclamar. Compone un email en un layout conformado por el email de contacto, el asunto y la foto del objeto.

Luego se codificó la acción para contactar en el controlador de objetos que hace el llamado al método definido anteriormente.

Contactar – Recibe como argumento el id del objeto a reclamar, encuentra el modelo y de este obtiene el email del usuario que lo reportó. Hace uso de un formulario de contacto incluido en el framework. Luego hace un llamado a la función de contacto que está en el modelo y envía el correo.

Finalmente se implementó la vista que se muestra en la bandeja de entrada del usuario.

Contactar – Contiene un formulario que consta del nombre de la persona que quiere reclamar el objeto, su email, el asunto, detalle y un código captcha de verificación.

4.2.3.5. Implementación de las seguridades de acceso

El framework permite la generación de reglas y restricciones por medio de una configuración que es instanciada en un json, la misma debe estar ubicada al inicio del controlador en la función behaviors.

Estas son las reglas del controlador de objetos, donde los usuarios públicos solamente pueden ver objetos y contactar, mientras que el administrador y los usuarios registrados, además de lo que se permite a los públicos, pueden administrar objetos (perdidos, en venta, vendidos y devueltos).

4.2.4. Pruebas

Una vez concluida la implementación, se prosiguió a hacer las pruebas para comprobar el correcto funcionamiento de la funcionalidad. Se hicieron las pruebas de caja blanca, caja negra y la de los criterios de aceptación.

4.2.4.1. Caja blanca

Haciendo uso de la base de datos creada para las pruebas y de las configuraciones efectuadas en el sprint anterior se hicieron las pruebas de aceptación correspondientes a esta funcionalidad. El código se comportó de manera satisfactoria.

4.2.4.2. Caja negra

De igual manera que en el anterior sprint se probaron valores límite para los atributos al momento de administrar objetos y contactar vía email. El sistema no presentó fallos en esta etapa con lo que se concluyó con éxito.

4.2.4.3. Criterios de aceptación

4.2.4.3.1. Despliegue del listado de objetos perdidos o en venta

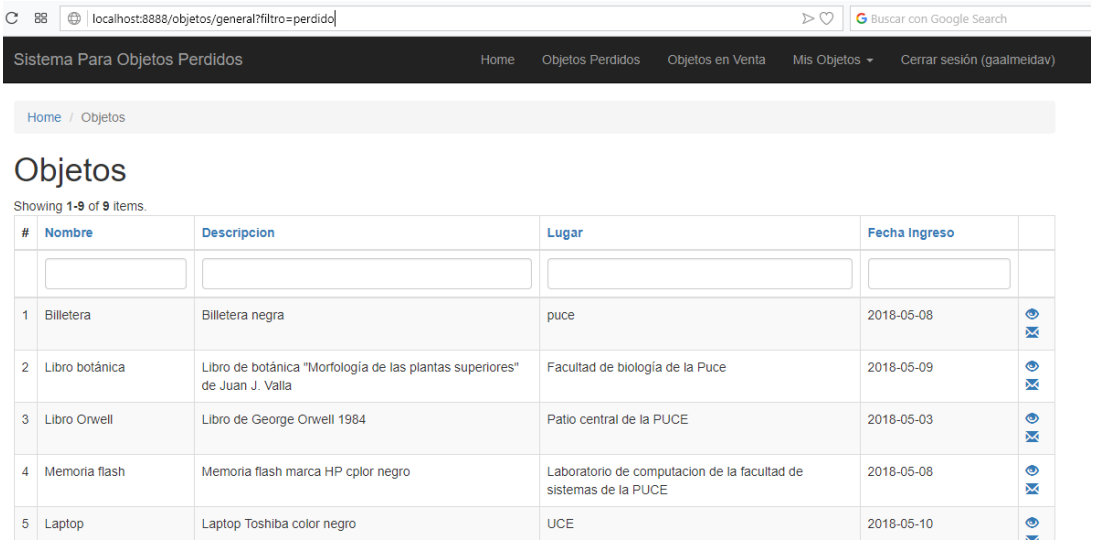

Ilustración 14 Listado de objetos perdidos o en venta

En el gráfico anterior situado en la barra de direcciones se puede ver el filtro de objetos perdidos, cambiándolo se puede acceder a los que se encuentran en venta.

4.2.4.3.2. Despliegue de los detalles de cada objeto

En el listado de objetos, al lado derecho del mismo se encuentra el ícono de ver el objeto individualmente. Al hacer click se pudo acceder a dicha vista.

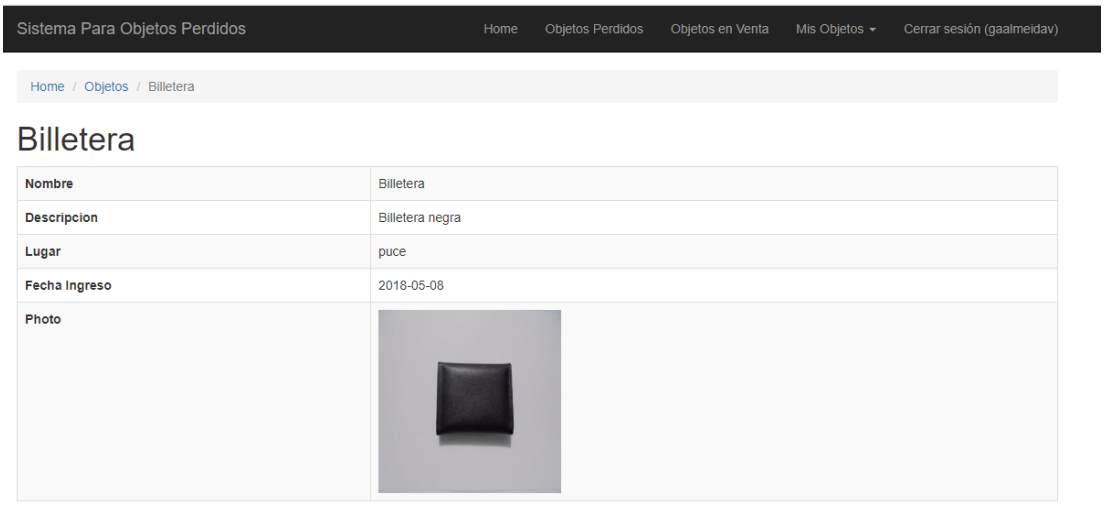

Ilustración 15 Vista de detalle de objeto

4.2.4.3.3. Envío de notificaciones vía email

Una vez llenado el formulario de contacto se constató que el correo fue recibido.

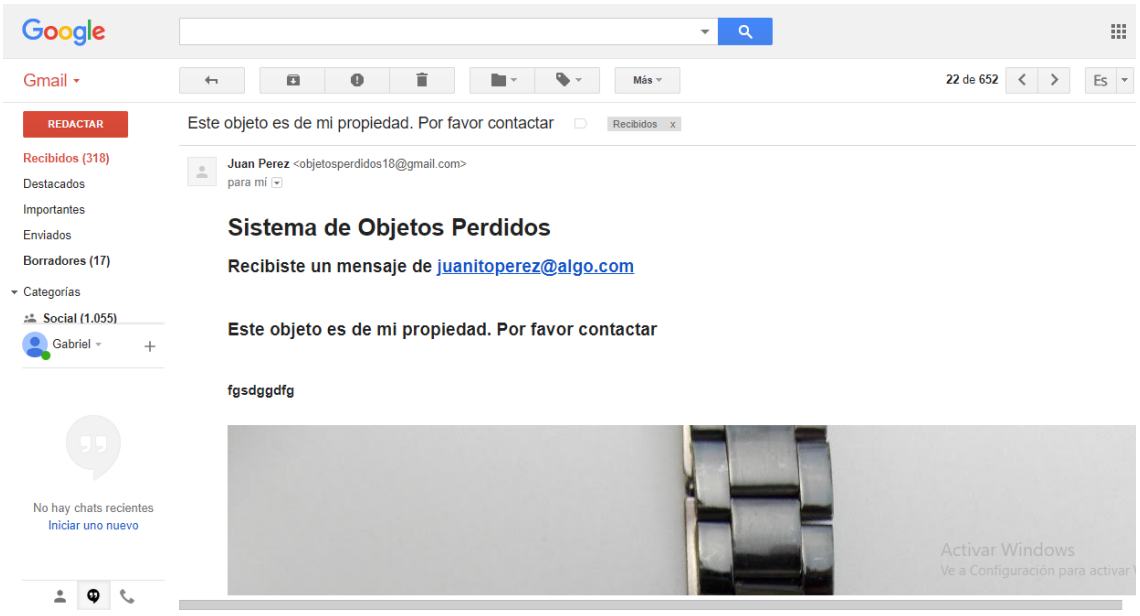

Ilustración 16 Vista de notificación por email

4.2.4.3.4. Ingreso de objetos al sistema

Se hizo la prueba de ingresar una memoria flash de 128 Gb encontrada en la Torre 1, piso 11 de la PUCE. Se constató el ingreso al ver dicho objeto en el listado general de objetos perdidos.

5	Laptop	Laptop Toshiba color negro	UCE	2018-05-10	
6	Laptop	Laptop Toshiba Satellite color negro y gris	Bar central de la PUCE	2018-05-15	
7	Libro	Libro de García Marquez Cien Años de Soledad	PUCE	2018-05-10	
8	Disco externo	Disco duro externo negro	Patio de la EPN	2018-05-03	
9	Flash memory	Flash de 128 gb	Torre 1 piso 11	2018-06-04	

Ilustración 17 Vista de objetos reportados

4.2.4.3.5. Despliegue del listado de objetos por estado

Un usuario registrado necesita acceder a sus objetos de acuerdo a su estado. Se ingresó desde el menú desplegable de administración de objetos.

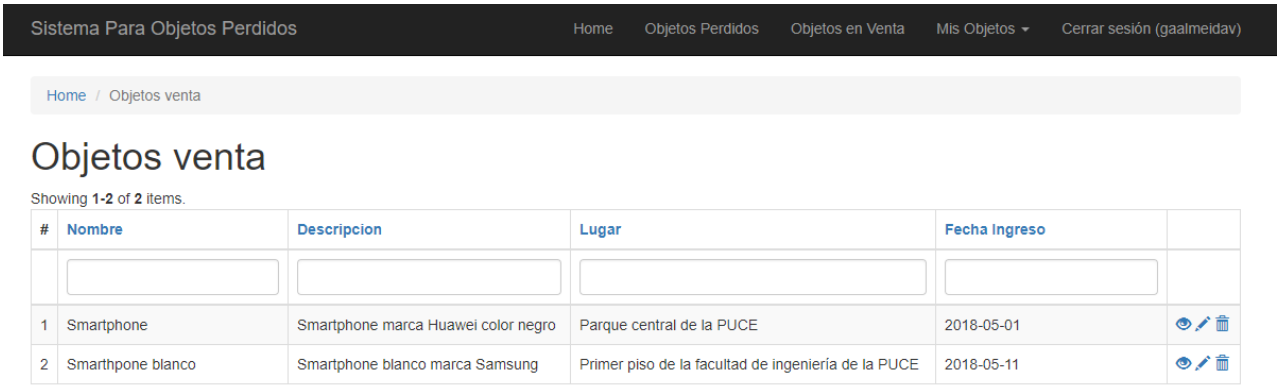

Ilustración 18 Vista de objetos por estado

4.2.4.3.6. Seguridades de acceso.

Se probaron las restricciones de acceso de los usuarios de acuerdo a su rol.

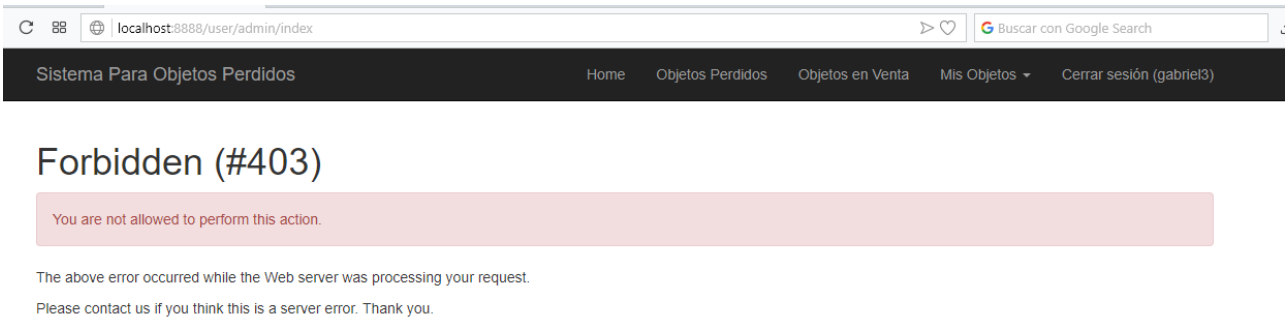

Ilustración 19 Vista restricción de acceso

5. Capítulo 5: Conclusiones y Recomendaciones

4.2. Conclusiones.

- La automatización de un sistema para objetos perdidos facilita la puesta en contacto entre quienes reportan y reclaman propiedad perdida.

- La metodología SCRUM permitió la adecuada administración de tareas y tiempos para lograr un desarrollo ágil de aplicaciones ya sean web, de escritorio, u otro tipo de arquitecturas.

- El framework Yii2 facilitó el desarrollo en todo el proyecto, manteniéndose en la arquitectura MVC. Además, hizo que la configuración de componentes sea un proceso más sencillo.

- Las herramientas de desarrollo proveyeron de todos los elementos necesarios para la construcción del aplicativo.

- La identificación de los roles para los usuarios hizo posible mantener las seguridades de acceso a la información contenida en el sistema.

4.3. Recomendaciones

- Identificar cuidadosamente los requerimientos del aplicativo para que este satisfaga al usuario final.

- Escoger una metodología de desarrollo que mejor se acople a los requerimientos levantados.

- Tratar de manejar SCRUM de la manera más formal posible para evitar reprogramación y reproceso de tareas.

- Seleccionar las herramientas tecnológicas que provean las funcionalidades básicas para un desarrollo eficiente.

- Escoger un framework adecuado que facilite la configuración y construcción de aplicativos.

- Implementar seguridades de acceso para proteger información privada en el sistema.

Bibliografía

Alegsa, L. (05 de diciembre, 2010). Definición de Sistema de Información. Recuperado de http://www.alegsa.com.ar/Dic/sistema_de_informacion.php

Alegsa, L. (22 de junio, 2016). Definición de Sistema Informático. Recuperado de http://www.alegsa.com.ar/Dic/sistema_informatico.php

Apache. (s.f.). HTTP Server Project. Recuperado de https://httpd.apache.org

Hernández, U. (24 de febrero, 2015). MVC explicado. Recuperado de https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

Roche, J. (29 de noviembre, 2017). Scrum: roles y responsabilidades. Recuperado de https://www2.deloitte.com/es/es/pages/technology/articles/roles-y-responsabilidades-scrum.html

Suyama, M. (30 de agosto, 2004). Protocolos de comunicaciones. Recuperado de https://desarrolloweb.com/articulos/1617.php

Mozilla. (16 de enero, 2018). ¿Cuál es la diferencia entre la página web, el sitio web, el servidor web y el motor de búsqueda? Recuperado de https://developer.mozilla.org/es/docs/Learn/Common_questions/Pages_sites_servers_and_search_engines

González, S. (21 de mayo, 2018). Generalidades del protocolo HTTP. Recuperado de https://developer.mozilla.org/es/docs/Web/HTTP/Overview

Mozilla. (24 de enero, 2018). Que es un servidor web? Recuperado de https://developer.mozilla.org/es/docs/Learn/Common_questions/Que_es_un_servidor_WEB

Valdes, K. (18 de marzo, 2016). What is a web browser? Recuperado de https://www.digitalunite.com/guides/using-internet-0/searching-browsing/what-web-browser

Oracle. (s.f.). Understanding Distributed Systems. Recuperado de https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SD173/ch1.htm

¿Qué es hardware y software? (s.f.). Recuperado de https://www.gcfaprendelibre.org/tecnologia/curso/informatica_basica/empezando_a_usar_un_computador/2.do

Adobe. (s.f.). Aspectos básicos de las aplicaciones web. Recuperado de https://helpx.adobe.com/mx/dreamweaver/using/web-applications.html

IBM. (s.f.). What is distributed computing. Recuperado dehttps://www.ibm.com/support/knowledgecenter/en/SSAL2T_8.2.0/com.ibm.cics.tx.doc/concepts/c_wht_is_distd_comptg.html

IBM. (s.f.). Three-tier architectures. Recuperado de https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/covr_3-tier.html

Roque, G. (29 de julio, 2014). Tipos de Software y su Clasificación. Recuperado de https://okhosting.com/blog/tipos-de-software-su-clasificacion/

Bailón, T. (30 de octubre, 2008). Qué es SCRUM. Recuperado de https://proyectosagiles.org/que-es-scrum/

Doshi, H. (4 de diciembre, 2016). The Three Pillars of Empiricism (Scrum). Recuperado de https://www.scrum.org/resources/blog/three-pillars-empiricism-scrum

Spear, S. (6 de abril, 2018). Scrum Values. Recuperado de https://www.scrumalliance.org/learn-about-scrum/scrum-values

Thakral, A. (19 de marzo, 2011). What is XAMPP? Recuperado de https://www.targetintegration.com/what-is-xampp/

Sass, R. (s.f.). What is a Web Browser. Recuperado de https://www.techopedia.com/definition/288/web-browser

Rouse, M. (19 de enero, 2015). ¿Qué es MySQL? Recuperado de https://searchdatacenter.techtarget.com/es/definicion/MySQL

Rouse, M. (15 de octubre, 2006). What is Firefox? Recuperado de https://searchmicroservices.techtarget.com/definition/Firefox

Pérez, J. (19 de agosto, 2015). ¿Qué es P2P? Recuperado de https://tecnologia-facil.com/que-es/que-es-p2p/

Torres, A. (28 de octubre, 2014). Software - Ciclo de Vida de Desarrollo. Recuperado de https://www.tutorialspoint.com/es/software_engineering/software_development_life_cycle.htm

Browser & Platform Market Share March 2018. (1 de abril, 2018). Recuperado de https://www.w3counter.com/globalstats.php

W3Schools. (14 de febrero, 2017). PHP 5 Introduction. Recuperado de https://www.w3schools.com/php/php_intro.asp

¿Qué es Yii? (26 de junio, 2014). Recuperado de https://yii2-framework.readthedocs.io/en/stable/guide-es/intro-yii/

Anexo A: Manual de Usuario

Usuario público:

El usuario público puede acceder al catálogo de objetos perdidos y en venta, además de contactar vía e-mail al usuario que ingresó el objeto en el sistema. Inicia ingresando al sitio:

Ilustración 20 Pantalla principal

A partir de ahí puede ingresar a las páginas de objetos perdidos o en venta, donde encontrará una lista de los objetos ahí presentes.

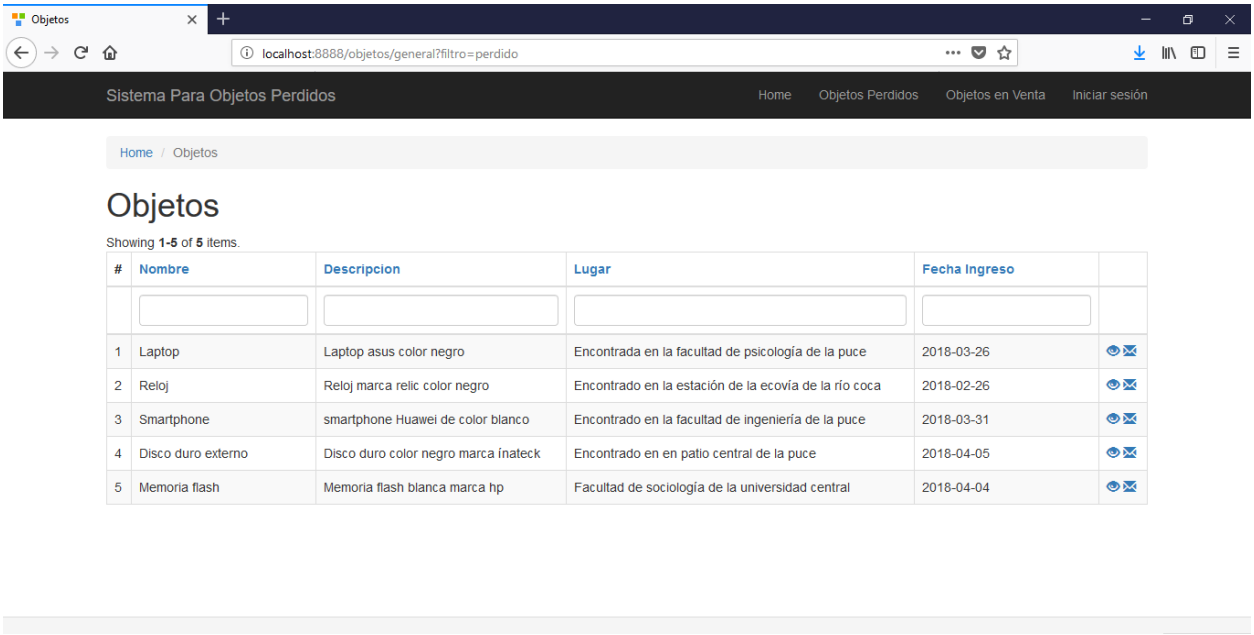

Ilustración 21 Pantalla de objetos perdidos

En la parte derecha de cada objeto se encuentran los íconos de ver y contactar. Al hacer click en ver se muestran los detalles del objeto.

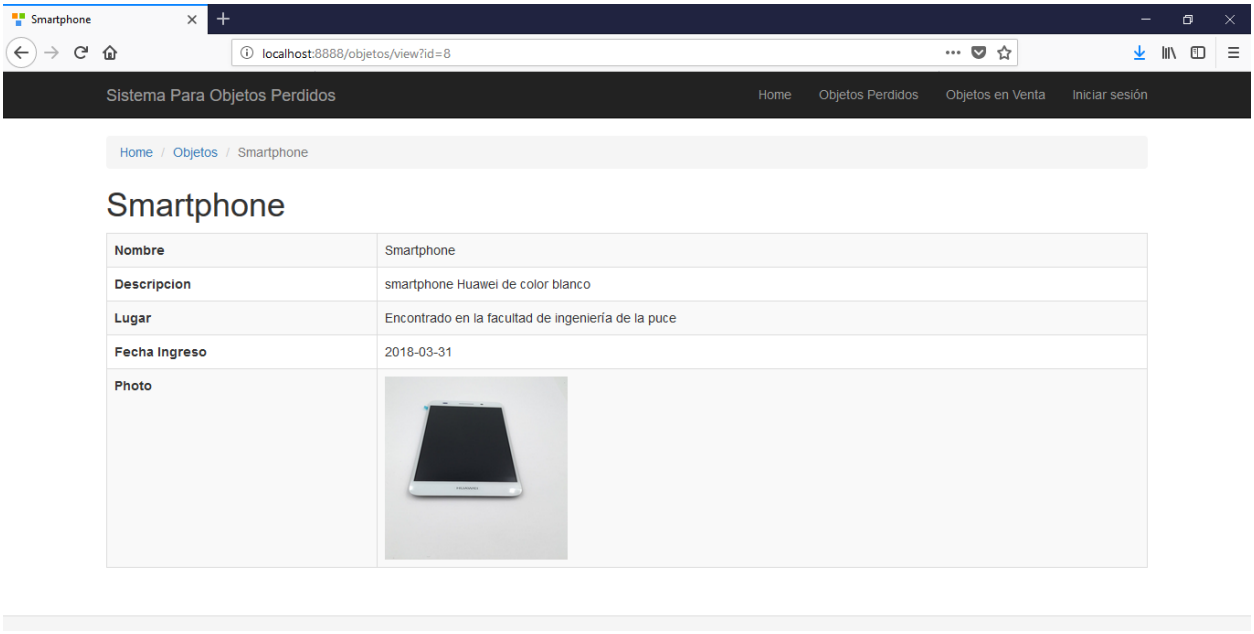

Ilustración 22 Pantalla de vista de objeto individual

En la ventana de contactar se muestra un formulario para enviar un email al usuario que reportó el objeto.

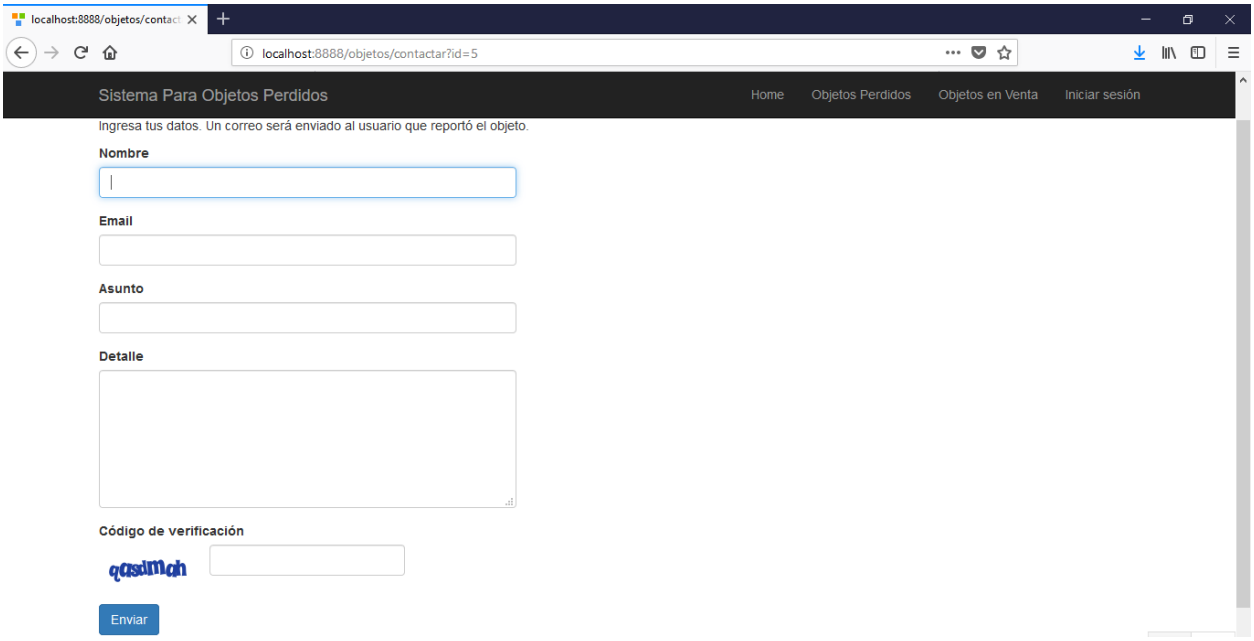

Ilustración 23 Pantalla de contacto

El usuario recibe un email con la información provista en el formulario anterior y la fotografía del objeto en cuestión.

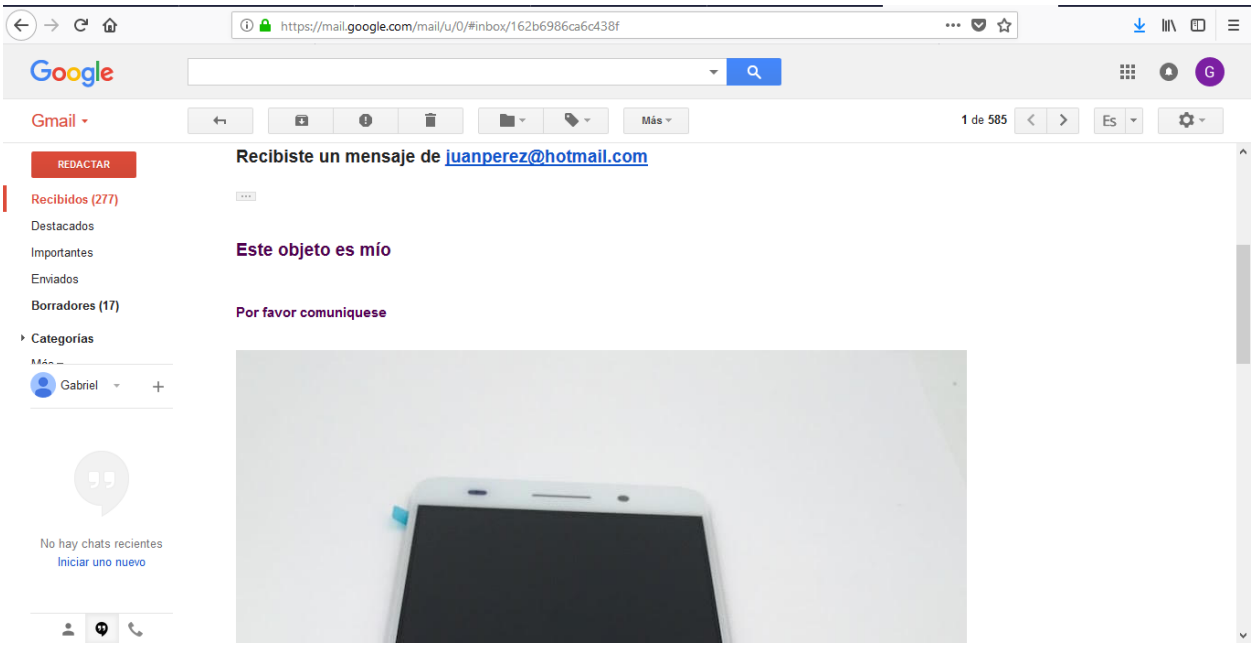

Ilustración 24 Correo electrónico recibido

Usuario registrado:

El usuario registrado puede hacer todo lo indicado anteriormente. Además de eso tiene la opción de administrar objetos. Inicia registrándose en el sistema.

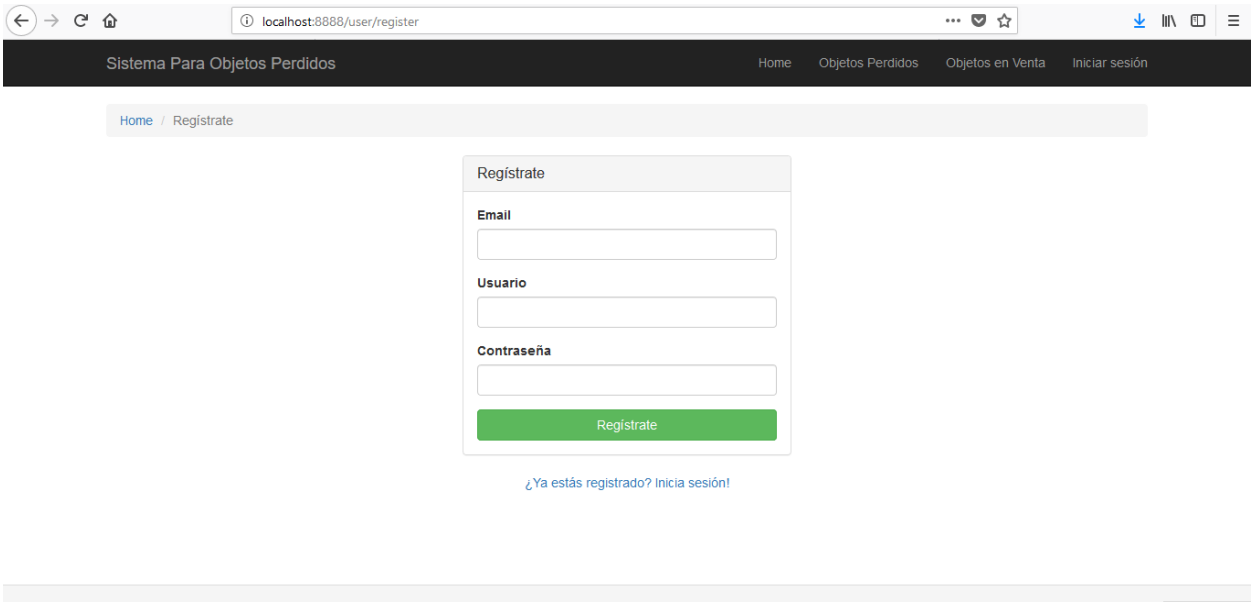

Ilustración 25 Pantalla de registro de usuarios

Una vez registrado aparece un menú desplegable para manejo de objetos. A partir de aquí puede administrar sus objetos reportados, devueltos, en venta y vendidos.

Ilustración 26 Pantalla principal con usuario registrado

Para reportar un objeto ingresa por la pestaña de “Nuevo Objeto”.

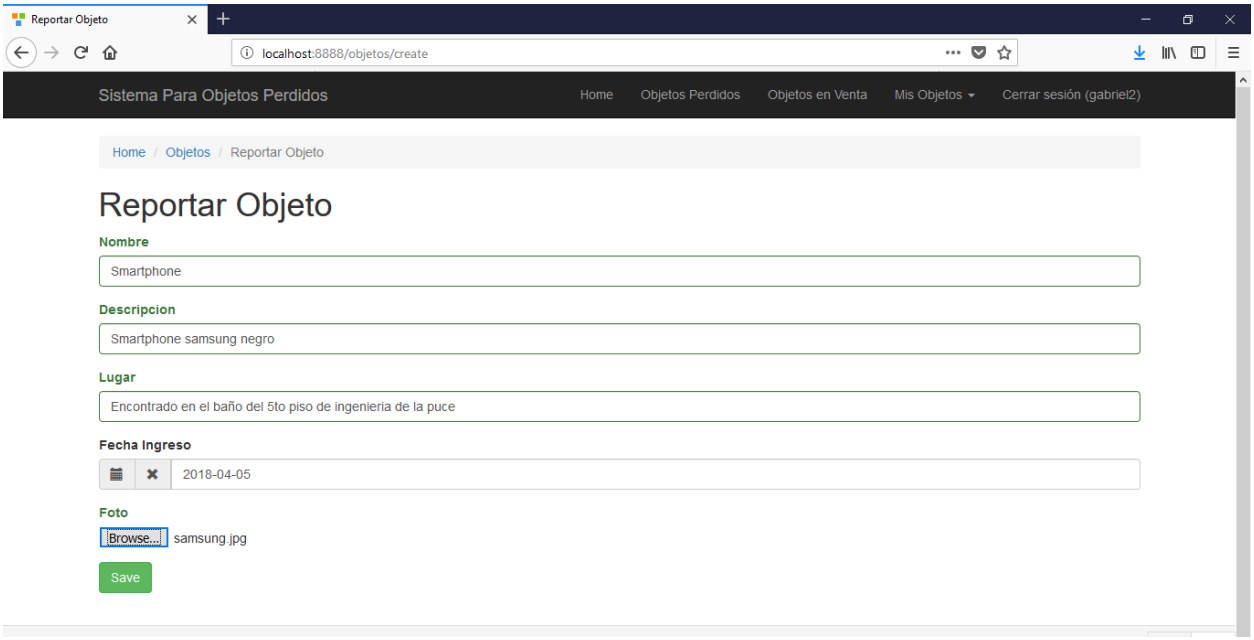

Ilustración 27 Pantalla de reporte de objeto perdido

EL usuario puede entonces ver los detalles del objeto ingresado, actualizarlo o eliminarlo.

Además, tiene la opción de devolver o poner en venta el objeto.

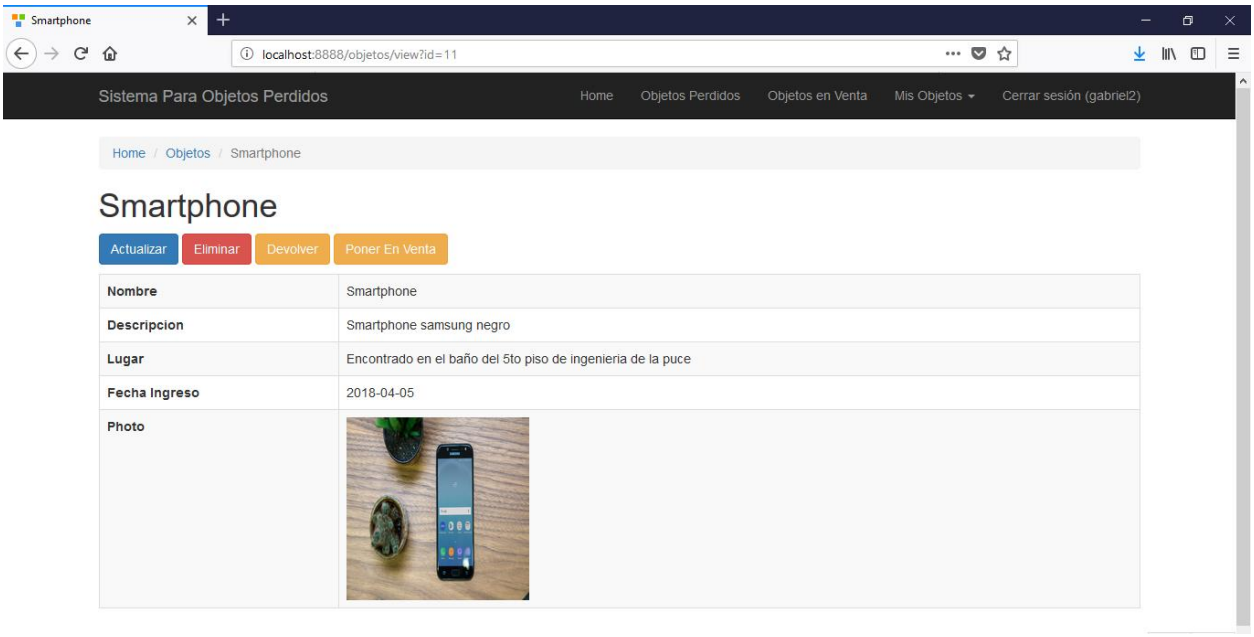

Ilustración 28 Pantalla de objeto individual con usuario registrado

Administrador:

El usuario administrador tiene acceso completo al sitio. Además de todo lo que pueden hacer los usuarios anteriores, tiene la capacidad de administrar usuarios.

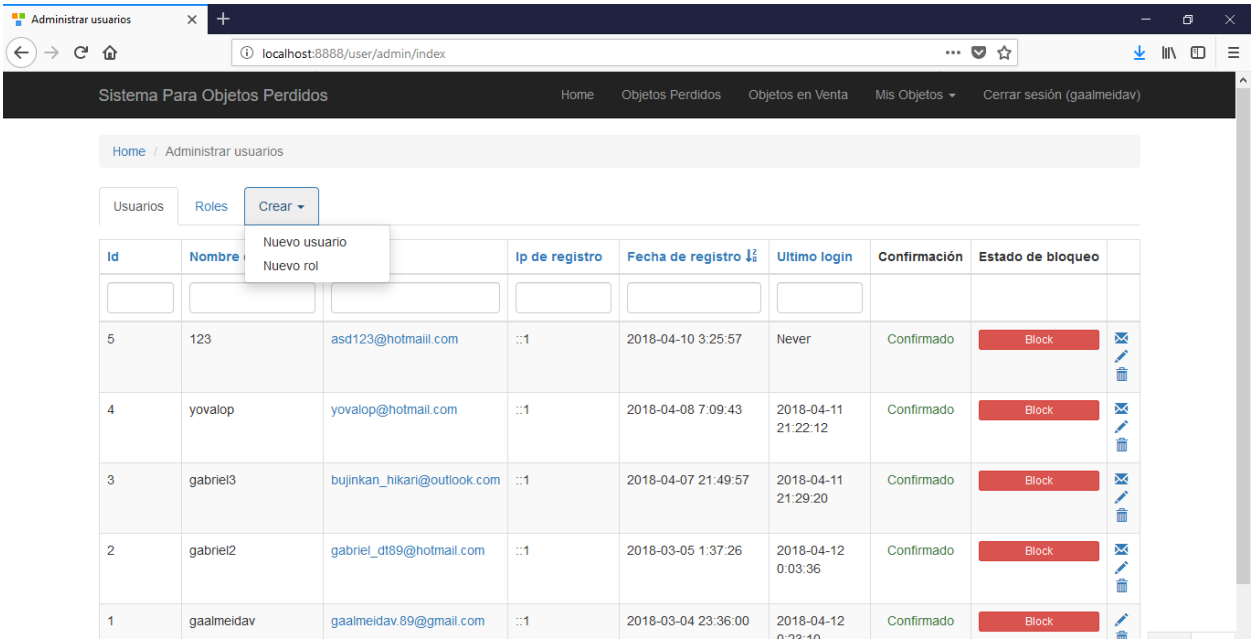

Ilustración 29 Pantalla de administración de usuarios

El administrador puede entonces manejar los detalles de la cuenta, del perfil y asignar roles a los usuarios.

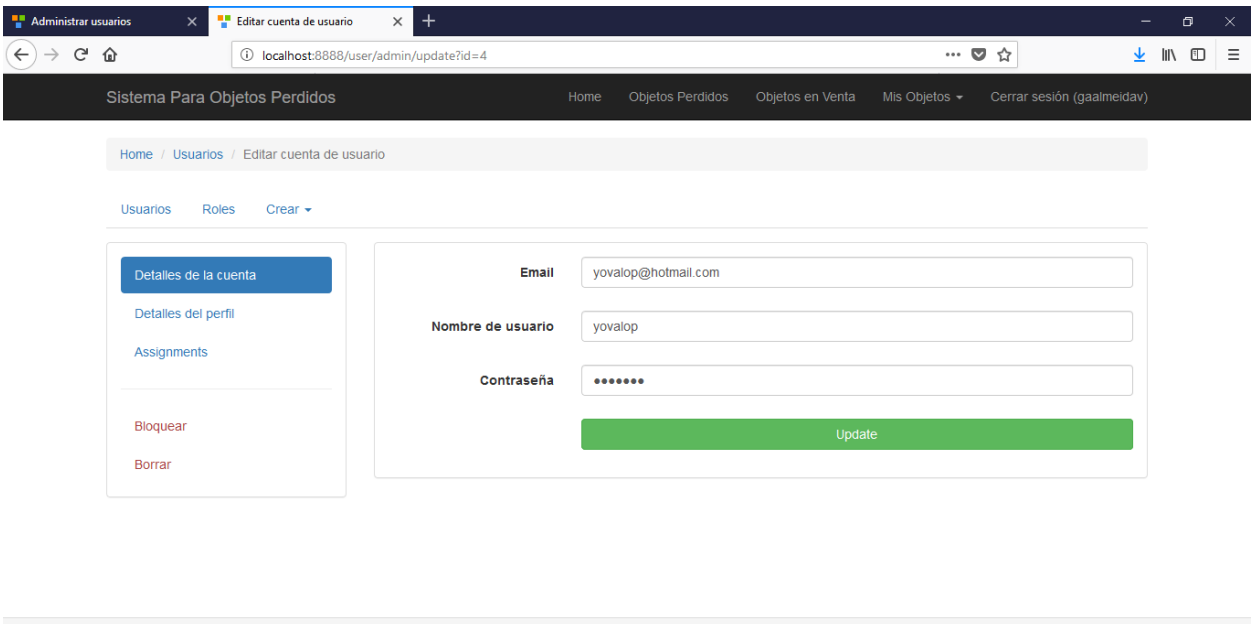

Ilustración 30 Pantalla de actualización de cuenta de usuario

Anexo B: Configuraciones

Instalación de la herramienta Xampp.

Puesto que el framework escogido, Yii2, trabaja con la herramienta Xampp, esta fue instalada a continuación. Primero descargó la aplicación del sitio web oficial: https://www.apachefriends.org/download.html. Una vez descargado se ejecutó el instalador; apareció la siguiente ventana:

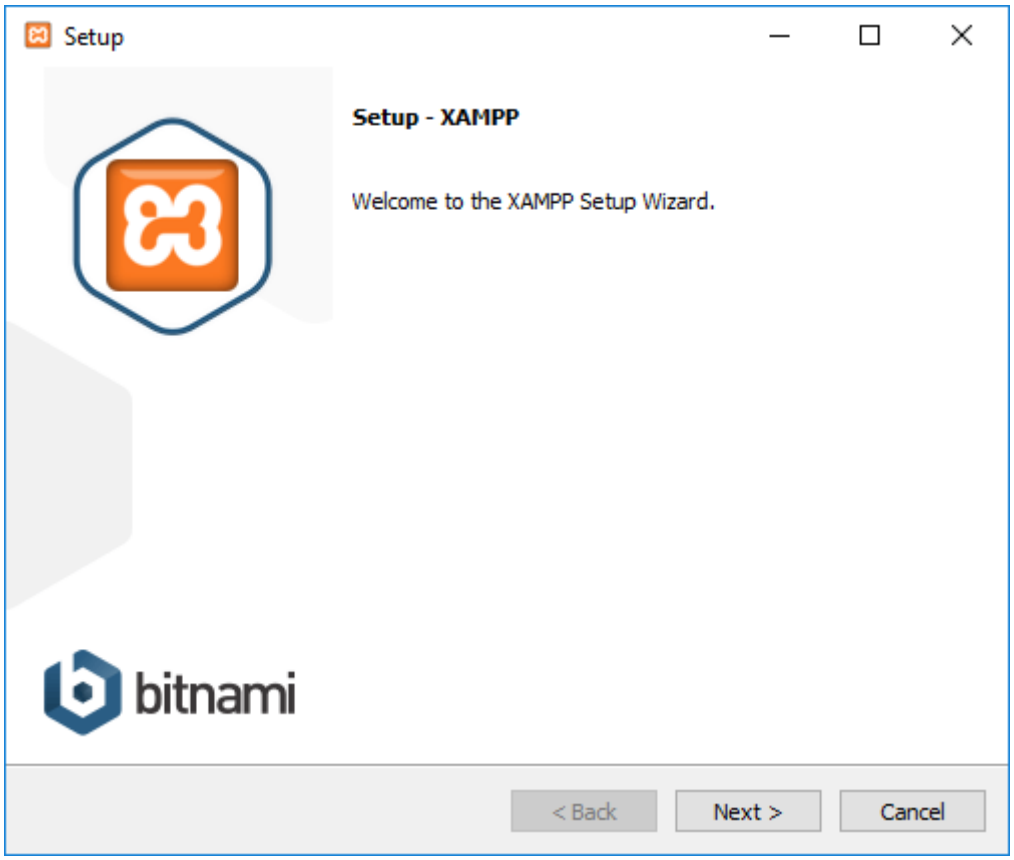

Ilustración 31 Setup Xampp

Al dar click en *Next* apareció la siguiente ventana en la que se escogieron los componentes de Xampp que se instalaron. Estos incluyen las herramientas del lado del servidor como Apache, MySQL, servidores FTP y de e-mails, los lenguajes PHP y Perl y la herramienta de administración

phpMyAdmin. Una vez escogidos, se prosiguió a la siguiente ventana, donde se especificó la ruta donde fue instalado Xampp en el sistema. A continuación, la herramienta se instaló y pudo ser ejecutada y utilizada:

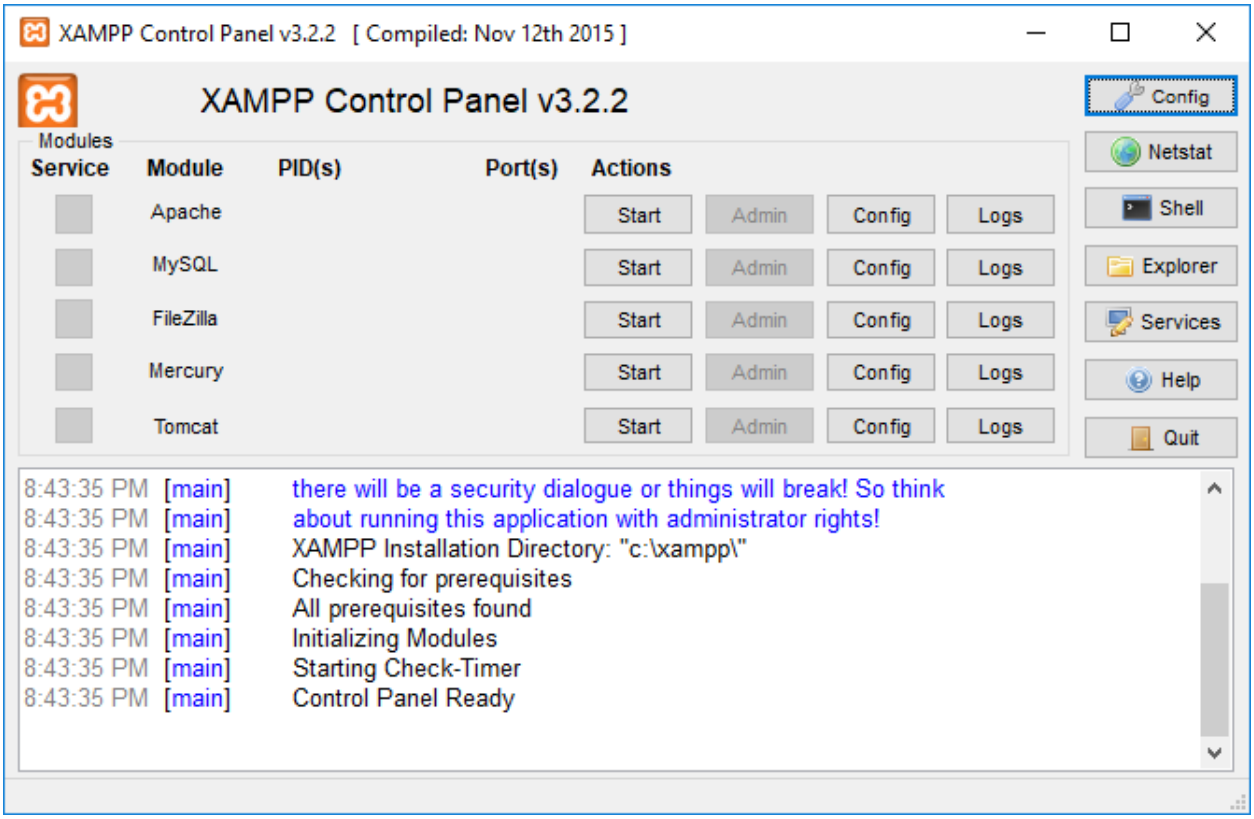

Ilustración 32 Panel de control de XAMPP

Para poder seguir con el siguiente paso, la creación de la base de datos, se iniciaron los módulos de Apache y MySQL. Luego se abrió el navegador web y en la barra de direcciones se ingresó http://localhost/phpmyadmin/index.php. Así se tuvo acceso a la herramienta administrativa phpMyAdmin:

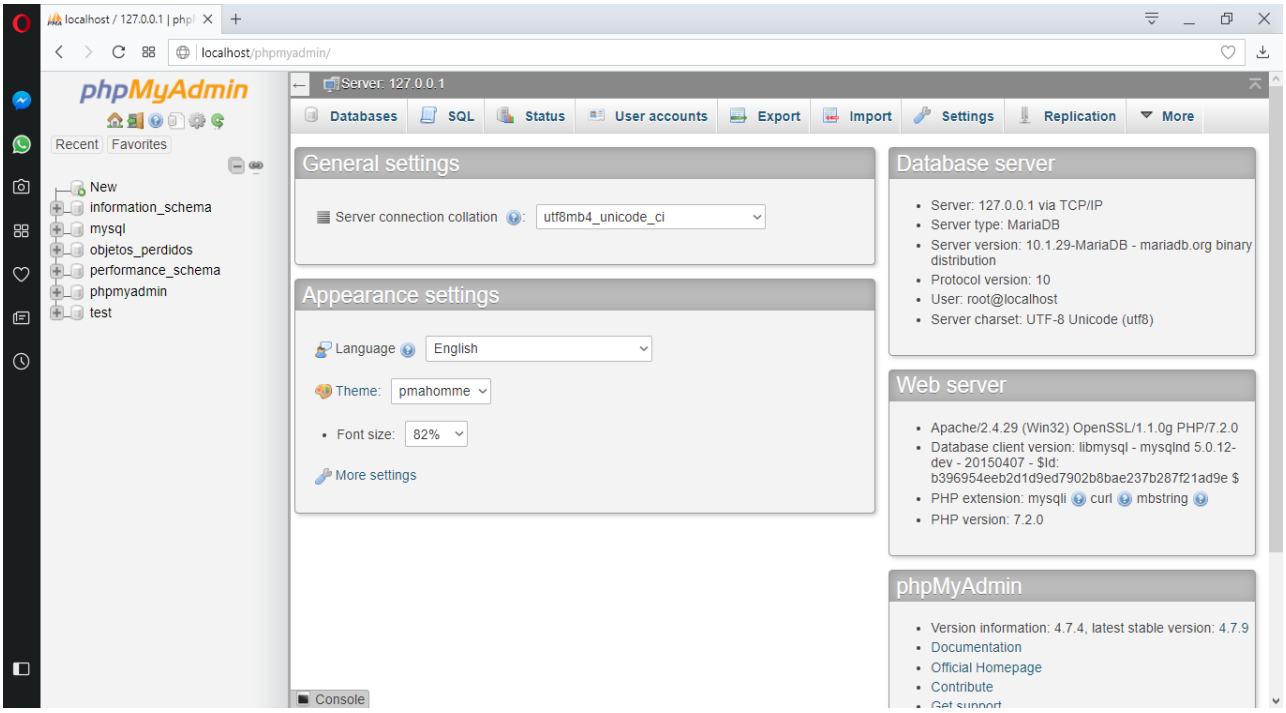

Ilustración 33 phpmyadmin

Instalación de la herramienta Composer.

Para poder instalar Yii2 se usó el manejador de dependencias de PHP, Composer. Se lo descargó de https://getcomposer.org.

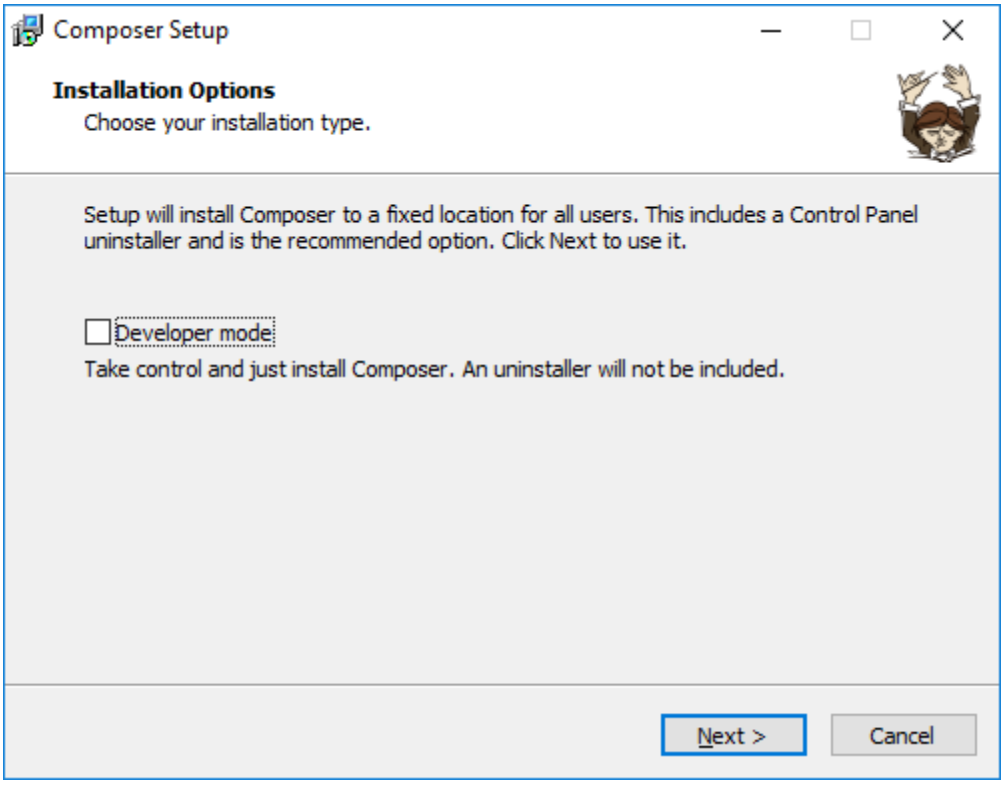

Ilustración 34 Instalación de Composer

Durante la instalación se especificó la línea de comandos de php que se usó. Además, ofreció la posibilidad de hacer uso de un servidor proxy para conectarse a internet.

Instalación de Yii2.

Una vez instalado Composer se procedió a instalar la herramienta Yii2. Se abrió la ventana de comandos y se ingresó al directorio htdocs de xampp. Se corrió el siguiente comando:

composer create-project --prefer-dist yiisoft/yii2-app-basic ObjetosProyecto.

Esto instaló una instancia de Yii2 en el directorio llamado ObjetosProyecto.

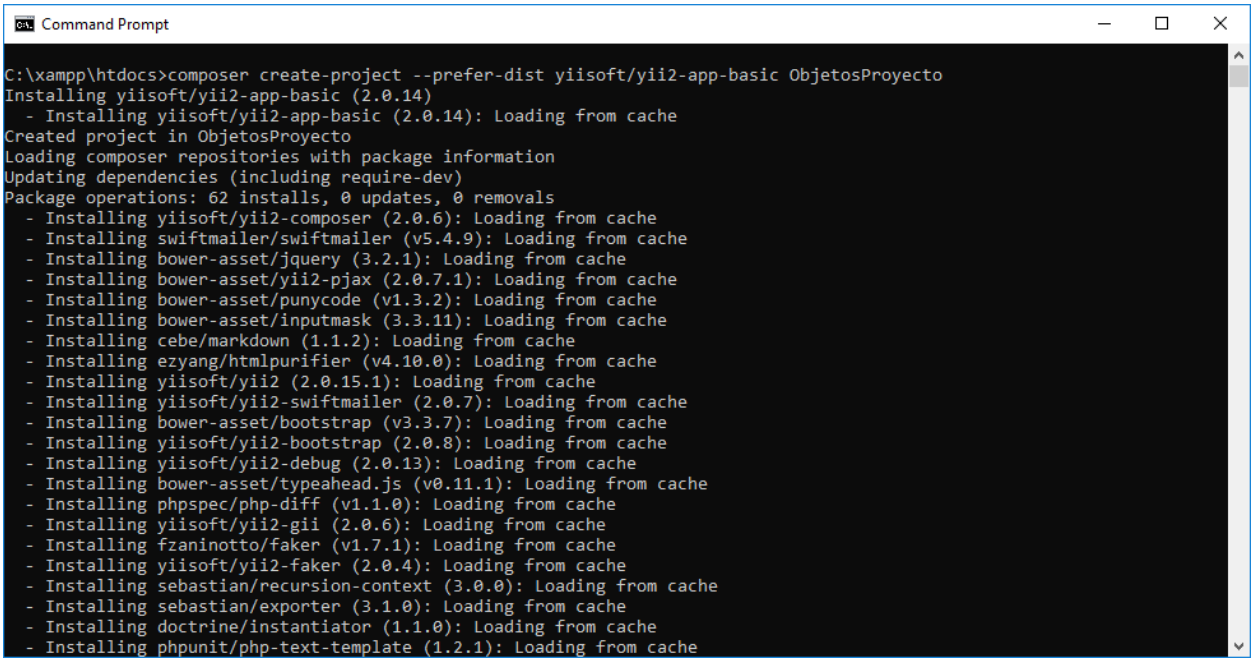

Ilustración 35 Ventana de comandos

El siguiente paso fue revisar si la instancia de Yii2 se instaló correctamente. En la ventana de comandos se ingresó al directorio ObjetosProyecto y se corrió el siguiente comando para iniciar el servidor:

php yii serve --port=8888.

Una vez hecho esto se procedió a abrir el navegador web e ir a *localhost* en el puerto 8888 (el número del puerto se escogió por facilidad).

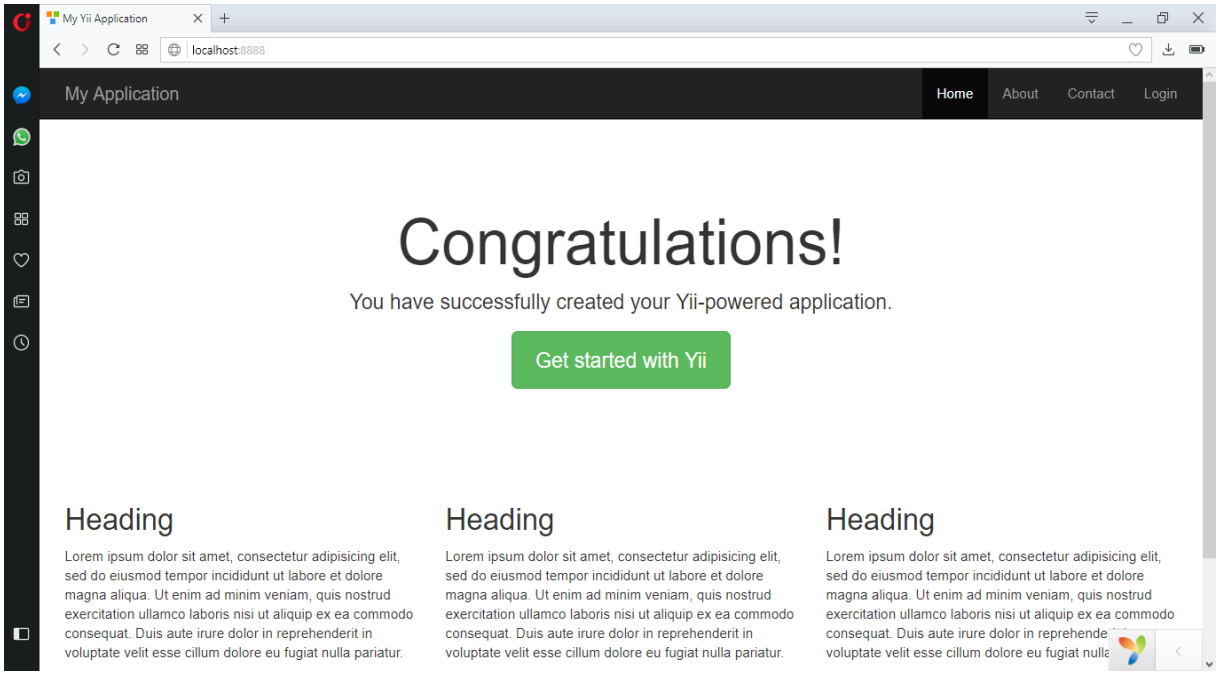

Ilustración 36 Pantalla de inicio

Anexo C: Código de la Implementación

Funcionalidad de usuarios

Implementación del modelo

Create

```php
public function create()
{

    $transaction = $this->getDb()->beginTransaction();

    try {
        $this->password = $this->password == null ? Password::generate(8) : $this->password;

        $this->trigger(self::BEFORE_CREATE);

        if (!$this->save()) {
            $transaction->rollBack();
            return false;
        }

        $this->confirm();

        $this->mailer->sendWelcomeMessage($this, null, true);
        $this->trigger(self::AFTER_CREATE);

        $transaction->commit();

        return true;
    } catch (\Exception $e) {
        $transaction->rollBack();
        \Yii::warning($e->getMessage());
        throw $e;
    }
}
```

Ilustración 37 Código de la función create para usuarios

Attribute Labels

```php
/** @inheritdoc */
public function attributeLabels()
{
    return [
        'username'          => \Yii::t('user', 'Nombre de usuario'),
        'email'             => \Yii::t('user', 'Email'),
        'registration_ip'   => \Yii::t('user', 'Ip de registro'),
        'unconfirmed_email' => \Yii::t('user', 'Nuevo email'),
        'password'          => \Yii::t('user', 'Contraseña'),
        'created_at'        => \Yii::t('user', 'Fecha de registro'),
        'last_login_at'     => \Yii::t('user', 'Ultimo login'),
        'confirmed_at'      => \Yii::t('user', 'Fecha de confirmación'),
    ];
}
```

Ilustración 38 Código de la función de etiquetas de los atributos para usuarios

Rules.

```php
public function rules()
{
    return [
        // username rules
        'usernameTrim'     => ['username', 'trim'],
        'usernameRequired' => ['username', 'required', 'on' => ['register', 'create', 'connect', 'update']],
        'usernameMatch'    => ['username', 'match', 'pattern' => static::$usernameRegexp],
        'usernameLength'   => ['username', 'string', 'min' => 3, 'max' => 255],
        'usernameUnique'   => [
            'username',
            'unique',
            'message' => \Yii::t('user', 'El nombre de usuario ya existe en el sistema')
        ],

        // email rules
        'emailTrim'     => ['email', 'trim'],
        'emailRequired' => ['email', 'required', 'on' => ['register', 'connect', 'create', 'update']],
        'emailPattern'  => ['email', 'email'],
        'emailLength'   => ['email', 'string', 'max' => 255],
        'emailUnique'   => [
            'email',
            'unique',
            'message' => \Yii::t('user', 'La dirección de correo ya existe en el sistema')
        ],

        // password rules
        'passwordRequired' => ['password', 'required', 'on' => ['register']],
        'passwordLength'   => ['password', 'string', 'min' => 6, 'max' => 72, 'on' => ['register', 'create']],
    ];
}
```

Ilustración 39 Código de las reglas de los atributos de los usuarios

Implementación del controlador

Create

```php
public function actionCreate()
{

    $user = \Yii::createObject([
        'class'    => User::className(),
        'scenario' => 'create',
    ]);
    $event = $this->getUserEvent($user);

    $this->performAjaxValidation($user);

    $this->trigger(self::EVENT_BEFORE_CREATE, $event);
    if ($user->load(\Yii::$app->request->post()) && $user->create()) {
        \Yii::$app->getSession()->setFlash('success', \Yii::t('user', 'Se ha creado un usuario'));
        $this->trigger(self::EVENT_AFTER_CREATE, $event);
        return $this->redirect(['update', 'id' => $user->id]);
    }

    return $this->render('create', [
        'user' => $user,
    ]);
}
```

Ilustración 40 Código del controlador de creación de usuarios

Index.

```php
public function actionIndex()
{
    Url::remember('', 'actions-redirect');
    $searchModel  = \Yii::createObject(UserSearch::className());
    $dataProvider = $searchModel->search(\Yii::$app->request->get());

    return $this->render('index', [
        'dataProvider' => $dataProvider,
        'searchModel'  => $searchModel,
    ]);
}
```

Ilustración 41 Código del controlador de index de usuarios

Update

```php
{
    Url::remember('', 'actions-redirect');
    $user = $this->findModel($id);
    $user->scenario = 'update';
    $event = $this->getUserEvent($user);

    $this->performAjaxValidation($user);

    $this->trigger(self::EVENT_BEFORE_UPDATE, $event);
    if ($user->load(\Yii::$app->request->post()) && $user->save()) {
        \Yii::$app->getSession()->setFlash('success', \Yii::t('user', 'Usuario actualizado'));
        $this->trigger(self::EVENT_AFTER_UPDATE, $event);
        return $this->refresh();
    }

    return $this->render('_account', [
        'user' => $user,
    ]);
}
```

Ilustración 42 Código del controlador de actualización de usuarios

Delete

```php
public function actionDelete($id)
{
    if ($id == \Yii::$app->user->getId()) {
        \Yii::$app->getSession()->setFlash('danger', \Yii::t('user', 'No puedes borrar tu propia cuenta'));
    } else {
        $model = $this->findModel($id);
        $event = $this->getUserEvent($model);
        $this->trigger(self::EVENT_BEFORE_DELETE, $event);
        $model->delete();
        $this->trigger(self::EVENT_AFTER_DELETE, $event);
        \Yii::$app->getSession()->setFlash('success', \Yii::t('user', 'El usuario se ha eliminado'));
    }

    return $this->redirect(['index']);
}
```

Ilustración 43 Código del controlador para borrar usuarios

Implementación de las vistas

Create

```php
$this->title = Yii::t('user', 'Crear una nueva cuenta de usuario');
$this->params['breadcrumbs'][] = ['label' => Yii::t('user', 'Users'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user'),]) ?>

<?= $this->render('_menu') ?>

<div class="row">
    <div class="col-md-3">
        <div class="panel panel-default">
            <div class="panel-body">
                <?= Nav::widget([
                    'options' => [
                        'class' => 'nav-pills nav-stacked',
                    ],
                    'items' => [
                        ['label' => Yii::t('user', 'Detalles de cuenta'), 'url' => ['/user/admin/create']],
                        ['label' => Yii::t('user', 'Detalles de perfil'), 'options' => [
                            'class' => 'disabled',
                            'onclick' => 'return false;',
                        ]],
                        ['label' => Yii::t('user', 'Informacion'), 'options' => [
                            'class' => 'disabled',
                            'onclick' => 'return false;',
                        ]],
                    ],
                ]) ?>
            </div>
        </div>
    </div>
    <div class="col-md-9">
        <div class="panel panel-default">
            <div class="panel-body">
                <div class="alert alert-info">
                    <?= Yii::t('user', 'Las credenciales serán enviadas al usuario por email') ?>.
                    <?= Yii::t('user', 'Una contraseña será generada automáticamente si no se especifica') ?>.
                </div>
                <?php $form = ActiveForm::begin([
                    'layout' => 'horizontal',
                    'enableAjaxValidation' => true
```

Ilustración 44 Código de la vista de creación de usuarios

Index

```php
$this->title = Yii::t('user', 'Administrar usuarios');
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user')]) ?>

<?= $this->render('/admin/_menu') ?>

<?php Pjax::begin() ?>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel'  => $searchModel,
    'layout'       => "{items}\n{pager}",
    'columns' => [
        [
            'attribute' => 'id',
            'headerOptions' => ['style' => 'width:90px;'], # 90px is sufficient for 5-digit user ids
        ],
        'username',
        'email:email',
        [
            'attribute' => 'registration_ip',
            'value' => function ($model) {
                return $model->registration_ip == null
                    ? '<span class="not-set">' . Yii::t('user', '(not set)') . '</span>'
                    : $model->registration_ip;
            },
            'format' => 'html',
        ],
        [
            'attribute' => 'created_at',
            'value' => function ($model) {
                if (extension_loaded('intl')) {
                    return Yii::t('user', '{0, date, MMMM dd, YYYY HH:mm}', [$model->created_at]);
                } else {
                    return date('Y-m-d G:i:s', $model->created_at);
                }
            },
        ],
```

Ilustración 45 Código de la vista de administración de usuarios

Update.

```php
$this->title = Yii::t('user', 'Editar cuenta de usuario');
$this->params['breadcrumbs'][] = ['label' => Yii::t('user', 'Usuarios'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user')]) ?>

<?= $this->render('_menu') ?>

<div class="row">
    <div class="col-md-3">
        <div class="panel panel-default">
            <div class="panel-body">
                <?= Nav::widget([
                    'options' => [
                        'class' => 'nav-pills nav-stacked',
                    ],
                    'items' => [
                        [
                            'label' => Yii::t('user', 'Detalles de la cuenta'),
                            'url' => ['/user/admin/update', 'id' => $user->id]
                        ],
                        [
                            'label' => Yii::t('user', 'Detalles del perfil'),
                            'url' => ['/user/admin/update-profile', 'id' => $user->id]
                        ],

                        '<hr>',
                        [
                            'label' => Yii::t('user', 'Confirmar'),
                            'url' => ['/user/admin/confirm', 'id' => $user->id],
                            'visible' => !$user->isConfirmed,
                            'linkOptions' => [
                                'class' => 'text-success',
                                'data-method' => 'post',
                                'data-confirm' => Yii::t('user', 'Estás seguro de querer confirmar este usuario?'),
                            ],
                        ],
                        [
                            'label' => Yii::t('user', 'Bloquear'),
```

Ilustración 46 Código de la vista de actualización de usuarios

Implementación del login

Modelo

Rules

```php
public function rules()
{
    $rules = [
        'loginTrim' => ['login', 'trim'],
        'requiredFields' => [['login'], 'required'],
        'confirmationValidate' => [
            'login',
            function ($attribute) {
                if ($this->user !== null) {
                    $confirmationRequired = $this->module->enableConfirmation
                        && !$this->module->enableUnconfirmedLogin;
                    if ($confirmationRequired && !$this->user->getIsConfirmed()) {
                        $this->addError($attribute, Yii::t('user', 'Necesitas confirmar tu dirección de correo'));
                    }
                    if ($this->user->getIsBlocked()) {
                        $this->addError($attribute, Yii::t('user', 'Tu cuenta ha sido bloqueada'));
                    }
                }
            }
        ],
        'rememberMe' => ['rememberMe', 'boolean'],
    ];

    if (!$this->module->debug) {
        $rules = array_merge($rules, [
            'requiredFields' => [['login', 'password'], 'required'],
            'passwordValidate' => [
                'password',
                function ($attribute) {
                    if ($this->user === null || !Password::validate($this->password, $this->user->password_hash)) {
                        $this->addError($attribute, Yii::t('user', 'Nombre de usuario o contraseña incorrecto'));
                    }
                }
            ]
        ]);
    }

    return $rules;
}
```

Ilustración 47 Código de la función rules del login

Validate Password

```php
 */
public function validatePassword($attribute, $params)
{
  if ($this->user === null || !Password::validate($this->password, $this->user->password_hash))
    $this->addError($attribute, Yii::t('user', 'Nombre de usuario o contraseña incorrecto'));
}
```

Ilustración 48 Código de validación de contraseña

Login

```php
public function login()
{
    if ($this->validate() && $this->user) {
        $isLogged = Yii::$app->getUser()->login($this->user, $this->rememberMe ? $this->module->rememberFor : 0);

        if ($isLogged) {
            $this->user->updateAttributes(['last_login_at' => time()]);
        }

        return $isLogged;
    }

    return false;
}
```

Ilustración 49 Código de la función de login

Controlador

Login

```php
public function actionLogin()
{
    if (!\Yii::$app->user->isGuest) {
        $this->goHome();
    }


    $model = \Yii::createObject(LoginForm::className());
    $event = $this->getFormEvent($model);

    $this->performAjaxValidation($model);

    $this->trigger(self::EVENT_BEFORE_LOGIN, $event);

    if ($model->load(\Yii::$app->getRequest()->post()) && $model->login()) {
        $this->trigger(self::EVENT_AFTER_LOGIN, $event);
        return $this->goBack();
    }

    return $this->render('login', [
        'model'  => $model,
        'module' => $this->module,
    ]);
}
```

Ilustración 50 Código del controlador para login

Logout

```php
    */
    public function actionLogout()
    {
        $event = $this->getUserEvent(\Yii::$app->user->identity);

        $this->trigger(self::EVENT_BEFORE_LOGOUT, $event);

        \Yii::$app->getUser()->logout();

        $this->trigger(self::EVENT_AFTER_LOGOUT, $event);

        return $this->goHome();
    }
```

Ilustración 51 Código del controlador para logout

Vista

Login

```php
$this->title = Yii::t('user', 'Iniciar sesión');
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user')]) ?>

<div class="row">
    <div class="col-md-4 col-md-offset-4 col-sm-6 col-sm-offset-3">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title"><?= Html::encode($this->title) ?></h3>
            </div>
            <div class="panel-body">
                <?php $form = ActiveForm::begin([
                    'id' => 'login-form',
                    'enableAjaxValidation' => true,
                    'enableClientValidation' => false,
                    'validateOnBlur' => false,
                    'validateOnType' => false,
                    'validateOnChange' => false,
                ]) ?>

                <?php if ($module->debug): ?>
                    <?= $form->field($model, 'login', [
                        'inputOptions' => [
                            'autofocus' => 'autofocus',
                            'class' => 'form-control',
                            'tabindex' => '1']])->dropDownList(LoginForm::loginList());
                    ?>

                <?php else: ?>

                    <?= $form->field($model, 'login',
                        ['inputOptions' => ['autofocus' => 'autofocus', 'class' => 'form-control', 'tabindex' => '1']]
                    );
                    ?>

                <?php endif ?>

                <?php if ($module->debug): ?>
                    <div class="alert alert-warning">
```

Ilustración 52 Código de la vista de login

Implementación del registro de usuarios

Modelo

Rules

```php
public function rules()
{
    $user = $this->module->modelMap['User'];

    return [
        // username rules
        'usernameTrim'     => ['username', 'trim'],
        'usernameLength'   => ['username', 'string', 'min' => 3, 'max' => 255],
        'usernamePattern'  => ['username', 'match', 'pattern' => $user::$usernameRegexp],
        'usernameRequired' => ['username', 'required'],
        'usernameUnique'   => [
            'username',
            'unique',
            'targetClass' => $user,
            'message' => Yii::t('user', 'Este nombre de usuario ya está registrado')
        ],
        // email rules
        'emailTrim'     => ['email', 'trim'],
        'emailRequired' => ['email', 'required'],
        'emailPattern'  => ['email', 'email'],
        'emailUnique'   => [
            'email',
            'unique',
            'targetClass' => $user,
            'message' => Yii::t('user', 'Esta dirección de email ya está registrada')
        ],
        // password rules
        'passwordRequired' => ['password', 'required', 'skipOnEmpty' => $this->module->enableGeneratingPassword],
        'passwordLength'   => ['password', 'string', 'min' => 6, 'max' => 72],
    ];
}
```

Ilustración 53 Código de las reglas del registro de usuarios

Register

```php
 */
public function register()
{
    if (!$this->validate()) {
        return false;
    }


    $user = Yii::createObject(User::className());
    $user->setScenario('register');
    $this->loadAttributes($user);

    if (!$user->register()) {
        return false;
    }

    Yii::$app->session->setFlash(
        'info',
        Yii::t(
            'user',
            'Se ha creado tu cuenta y un mensaje con instrucciones
            adicionales ha sido enviado a tu correo'
        )
    );

    return true;
}
```

Ilustración 54 Código de la función registrar

Controlador

Register

```php
 */
public function actionRegister()
{
    if (!$this->module->enableRegistration) {
        throw new NotFoundHttpException();
    }

    $model = \Yii::createObject(RegistrationForm::className());
    $event = $this->getFormEvent($model);

    $this->trigger(self::EVENT_BEFORE_REGISTER, $event);

    $this->performAjaxValidation($model);

    if ($model->load(\Yii::$app->request->post()) && $model->register()) {
        $this->trigger(self::EVENT_AFTER_REGISTER, $event);

        return $this->render('/message', [
            'title'  => \Yii::t('user', 'Se ha creado tu cuenta'),
            'module' => $this->module,
        ]);
    }

    return $this->render('register', [
        'model'  => $model,
        'module' => $this->module,
    ]);
}
```

Ilustración 55 Código del controlador para registro

Vista

Register

```php
$this->title = Yii::t('user', 'Regístrate');
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="row">
    <div class="col-md-4 col-md-offset-4 col-sm-6 col-sm-offset-3">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title"><?= Html::encode($this->title) ?></h3>
            </div>
            <div class="panel-body">
                <?php $form = ActiveForm::begin([
                    'id' => 'registration-form',
                    'enableAjaxValidation' => true,
                    'enableClientValidation' => false,
                ]); ?>

                <?= $form->field($model, 'email') ?>

                <?= $form->field($model, 'username') ?>

                <?php if ($module->enableGeneratingPassword == false): ?>
                    <?= $form->field($model, 'password')->passwordInput() ?>
                <?php endif ?>

                <?= Html::submitButton(Yii::t('user', 'Regístrate'), ['class' => 'btn btn-success btn-block']) ?>

                <?php ActiveForm::end(); ?>
            </div>
        </div>
        <p class="text-center">
            <?= Html::a(Yii::t('user', '¿Ya estás registrado? Inicia sesión!'), ['/user/security/login']) ?>
        </p>
    </div>
</div>
```

Ilustración 56 Código de la vista de registro

Implementación de perfiles de usuario

Controlador

Index

```php
61      public function actionIndex()
62      {
63          $filterModel = new Search($this->type);
64          return $this->render('index', [
65              'filterModel'  => $filterModel,
66              'dataProvider' => $filterModel->search(\Yii::$app->request->get()),
67          ]);
68      }
```

Ilustración 57 Código del controlador de index para roles

Create

```php
public function actionCreate()
{

    $model = \Yii::createObject([
        'class'    => $this->modelClass,
        'scenario' => 'create',
    ]);

    $this->performAjaxValidation($model);

    if ($model->load(\Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['index']);
    }

    return $this->render('create', [
        'model' => $model,
    ]);
}

```

Ilustración 58 Código del controlador de creación de roles

Update

```php
public function actionUpdate($name)
{

    $item  = $this->getItem($name);
    $model = \Yii::createObject([
        'class'    => $this->modelClass,
        'scenario' => 'update',
        'item'     => $item,
    ]);

    $this->performAjaxValidation($model);

    if ($model->load(\Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['index']);
    }

    return $this->render('update', [
        'model' => $model,
    ]);
}

```

Ilustración 59 Código del controlador de actualización de roles

Delete

```php
    public function actionDelete($name)
    {
        $item = $this->getItem($name);
        \Yii::$app->authManager->remove($item);
        return $this->redirect(['index']);
    }

```

Ilustración 60 Código del controlador de eliminación de roles

Vista

Index

```php
$this->title = Yii::t('user', 'Administrar usuarios');
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user')]) ?>

<?= $this->render('/admin/_menu') ?>

<?php Pjax::begin() ?>

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'filterModel'  => $searchModel,
    'layout'       => "{items}\n{pager}",
    'columns' => [
        [
            'attribute' => 'id',
            'headerOptions' => ['style' => 'width:90px;'], # 90px is sufficient for 5-digit user ids
        ],
        'username',
        'email:email',
        [
            'attribute' => 'registration_ip',
            'value' => function ($model) {
                return $model->registration_ip == null
                    ? '<span class="not-set">' . Yii::t('user', '(not set)') . '</span>'
                    : $model->registration_ip;
            },
            'format' => 'html',
        ],
        [
            'attribute' => 'created_at',
            'value' => function ($model) {
                if (extension_loaded('intl')) {
                    return Yii::t('user', '{0, date, MMMM dd, YYYY HH:mm}', [$model->created_at]);
                } else {
                    return date('Y-m-d G:i:s', $model->created_at);
                }
            },
        ],
```

Ilustración 61 Código de la vista de index para roles

Create

```php

$this->title = Yii::t('user', 'Crear una nueva cuenta de usuario');
$this->params['breadcrumbs'][] = ['label' => Yii::t('user', 'Users'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user'),]) ?>

<?= $this->render('_menu') ?>

<div class="row">
    <div class="col-md-3">
        <div class="panel panel-default">
            <div class="panel-body">
                <?= Nav::widget([
                    'options' => [
                        'class' => 'nav-pills nav-stacked',
                    ],
                    'items' => [
                        ['label' => Yii::t('user', 'Detalles de cuenta'), 'url' => ['/user/admin/create']],
                        ['label' => Yii::t('user', 'Detalles de perfil'), 'options' => [
                            'class' => 'disabled',
                            'onclick' => 'return false;',
                        ]],
                        ['label' => Yii::t('user', 'Informacion'), 'options' => [
                            'class' => 'disabled',
                            'onclick' => 'return false;',
                        ]],
                    ],
                ]) ?>
            </div>
        </div>
    </div>
    <div class="col-md-9">
        <div class="panel panel-default">
            <div class="panel-body">
                <div class="alert alert-info">
                    <?= Yii::t('user', 'Las credenciales serán enviadas al usuario por email') ?>.
                    <?= Yii::t('user', 'Una contraseña será generada automáticamente si no se especifica') ?>.
                </div>
```

Ilustración 62 Código de la vista de creación para roles

Update

```php
$this->title = Yii::t('user', 'Editar cuenta de usuario');
$this->params['breadcrumbs'][] = ['label' => Yii::t('user', 'Usuarios'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

<?= $this->render('/_alert', ['module' => Yii::$app->getModule('user')]) ?>

<?= $this->render('_menu') ?>

<div class="row">
    <div class="col-md-3">
        <div class="panel panel-default">
            <div class="panel-body">
                <?= Nav::widget([
                    'options' => [
                        'class' => 'nav-pills nav-stacked',
                    ],
                    'items' => [
                        [
                            'label' => Yii::t('user', 'Detalles de la cuenta'),
                            'url' => ['/user/admin/update', 'id' => $user->id]
                        ],
                        [
                            'label' => Yii::t('user', 'Detalles del perfil'),
                            'url' => ['/user/admin/update-profile', 'id' => $user->id]
                        ],
                        [
                            'label' => Yii::t('user', 'Assignments'),
                            'url' => ['/user/admin/assignments', 'id' => $user->id],
                            'visible' => isset(Yii::$app->extensions['dektrium/yii2-rbac']),
                        ],

                        '<hr>',
                        [
                            'label' => Yii::t('user', 'Confirmar'),
                            'url' => ['/user/admin/confirm', 'id' => $user->id],
                            'visible' => !$user->isConfirmed,
                            'linkOptions' => [
                                'class' => 'text-success',
                                'data-method' => 'post',
                                'data-confirm' => Yii::t('user', 'Estás seguro de querer confirmar este usuario?')
```

Ilustración 63 Código de la vista de actualización de roles

Funcionalidad de objetos

Implementación del modelo

Rules

```php
 */
public function rules()
{
    return [
        [['nombre', 'descripcion', 'lugar', 'foto', 'estado', 'us_id'], 'required'],
        [['fecha_encontrado', 'fecha_ingreso', 'fecha_vendido'], 'safe'],
        [['precio'], 'number'],
        [['estado'], 'string'],
        [['us_id'], 'integer'],
        [['nombre', 'descripcion', 'lugar', 'duenio', 'foto', 'comprador'], 'string', 'max' => 255],
    ];
}
```

Ilustración 64 Código de las reglas de los atributos de objetos

Attribute Labels

```php
public function attributeLabels()
{
    return [
        'id' => 'ID',
        'nombre' => 'Nombre',
        'descripcion' => 'Descripcion',
        'lugar' => 'Lugar',
        'fecha_encontrado' => 'Fecha Encontrado',
        'fecha_ingreso' => 'Fecha Ingreso',
        'duenio' => 'Dueño',
        'foto' => 'Foto',
        'precio' => 'Precio',
        'comprador' => 'Comprador',
        'fecha_vendido' => 'Fecha Vendido',
        'estado' => 'Estado',
        'us_id' => 'Us ID',
    ];
}
```

Ilustración 65 Código de las etiquetas de los atributos de objetos

Búsqueda dinámica

```php
public function search($params)
{
    $query = Objetos::find();


    $dataProvider = new ActiveDataProvider([
        'query' => $query,
    ]);

    $this->load($params);

    if (!$this->validate()) {

        return $dataProvider;
    }


    $query->andFilterWhere([
        'id' => $this->id,
        'fecha_encontrado' => $this->fecha_encontrado,
        'fecha_ingreso' => $this->fecha_ingreso,
        'precio' => $this->precio,
        'fecha_vendido' => $this->fecha_vendido,
    ]);

    $query->andFilterWhere(['like', 'nombre', $this->nombre])
        ->andFilterWhere(['like', 'descripcion', $this->descripcion])
        ->andFilterWhere(['like', 'lugar', $this->lugar])
        ->andFilterWhere(['like', 'duenio', $this->duenio])
        ->andFilterWhere(['like', 'foto', $this->foto])
        ->andFilterWhere(['like', 'comprador', $this->comprador]);

    return $dataProvider;
}
```

Ilustración 66 Código de búsqueda dinámica de objetos

Implementación del controlador

General

```php
public function actionGeneral($filtro)
{
    $searchModel = new ObjetosSearch();
    $dataProvider = new ActiveDataProvider([
        'query' => Objetos::find()->Where(['estado' => $filtro]),
    ]);

    return $this->render('general', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
    ]);
}
```

Ilustración 67 Código del controlador general de objetos

Index

```php
public function actionIndex($filtro)
{
    $searchModel = new ObjetosSearch();
    $dataProvider = new ActiveDataProvider([
        'query' => Objetos::find()->where(['us_id'=> Yii::$app->user->identity->id])->andWhere(['estado' => $filtro])
            ,
    ]);

    return $this->render('index', [
        'searchModel' => $searchModel,
        'dataProvider' => $dataProvider,
        'titulo' => $filtro,
    ]);
}
```

Ilustración 68 Código del controlador del index de objetos

View

```php
public function actionView($id)
{
    return $this->render('view', [
        'model' => $this->findModel($id),
    ]);
}
```

Ilustración 69 Código del controlador de vista de objetos

Create

```php
public function actionCreate()
{
    $model = new Usuarios();

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->usu_id]);
    }

    return $this->render('create', [
        'model' => $model,
    ]);
}

/**
```

Ilustración 70 Código del controlador de creación de usuarios

Update

```php
public function actionUpdate($id)
{
    $model = $this->findModel($id);

    if ($model->load(Yii::$app->request->post()) && $model->save()) {
        return $this->redirect(['view', 'id' => $model->usu_id]);
    }

    return $this->render('update', [
        'model' => $model,
    ]);
}
```

Ilustración 71 Código del controlador de actualización de objetos

Delete

```php
/**
public function actionDelete($id)
{
    $this->findModel($id)->delete();

    return $this->redirect(['index']);
}
```

Ilustración 72 Código del controlador de borrado de objetos

Actualizar

```php
public function actionActualizar($id, $estado)
{
    $model = $this->findModel($id);
    $model->estado = $estado;

    switch ($model->estado) {
        case 'perdido':
            $botonGuardar = "Devolver Objeto";
            break;

        case 'venta':
            $botonGuardar = "Poner en Venta Objeto";
            break;

        case 'vendido':
            $botonGuardar = "Vender Objeto";
            break;

        default:
            throw new \yii\web\HttpException(404, 'The requested Item could not be found.');
            break;
    }

    if ($model->load(Yii::$app->request->post())) {
        if($model->estado === "perdido")
            $model->estado = "devuelto";

        $model->save();

        return $this->redirect(['view', 'id' => $model->id]);
    }



    return $this->render('actualizar', [
        'model' => $model,
        'botonGuardar' => $botonGuardar,
    ]);
}
```

Ilustración 73 Código del controlador de actualización del estado de objetos

Implementación de las vistas

Create

```php
<?php

use yii\helpers\Html;



$this->title = 'Reportar Objeto';
$this->params['breadcrumbs'][] = ['label' => 'Objetos', 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="objetos-create">

    <h1><?= Html::encode($this->title) ?></h1>

    <?= $this->render('_form', [
        'model' => $model,
    ]) ?>

</div>
```

Ilustración 74 Código de la vista de creación de objetos

Update

```php
<?php

use yii\helpers\Html;


$this->title = 'Editar objeto';
$this->params['breadcrumbs'][] = ['label' => 'Objetos', 'url' => ['index']];
$this->params['breadcrumbs'][] = ['label' => $model->id, 'url' => ['view', 'id' => $model->id]];
$this->params['breadcrumbs'][] = 'Update';
?>
<div class="objetos-update">

    <h1><?= Html::encode($this->title) ?></h1>

    <?= $this->render('_form', [
        'model' => $model,
    ]) ?>

</div>
```

Ilustración 75 Código de la vista de actualización de objetos

Formulario para las vistas de creación y edición

```php
<div class="objetos-form">

    <?php $form = ActiveForm::begin(); ?>

    <?= ($model->estado == '' || $model->estado == 'perdido') ? $form->field($model, 'nombre')->textInput(['maxlength' =>
        true]) : "" ?>

    <?= ($model->estado == '' || $model->estado == 'perdido') ?  $form->field($model, 'descripcion')->textInput(['
        maxlength' => true]) : "" ?>

    <?= ($model->estado == '' || $model->estado == 'perdido') ? $form->field($model, 'lugar')->textInput(['maxlength' =>
        true]) : "" ?>

    <?= $model->estado == 'devuelto' ? $form->field($model, 'fecha_encontrado')->widget(DatePicker::classname(), [
        'options' => ['placeholder' => 'Ingrese Fecha ...'],
        'pluginOptions' => [
            'autoclose'=>true,
            'format' => 'yyyy-mm-dd'
        ]
    ]) : "" ?>

    <?= ($model->estado == '' || $model->estado == 'perdido') ? $form->field($model, 'fecha_ingreso')->widget(DatePicker::
        classname(), [
        'options' => ['placeholder' => 'Ingrese Fecha ...'],
        'pluginOptions' => [
            'autoclose'=>true,
            'format' => 'yyyy-mm-dd'
        ]
    ]) : "" ?>

    <?= $model->estado == 'devuelto' ? $form->field($model, 'duenio')->textInput(['maxlength' => true]) :"" ?>

    <?= ($model->estado == '' || $model->estado == 'perdido') ?  $form->field($model, 'foto')->fileInput() : "" ?>

    <?= $model->estado == 'venta' ? $form->field($model, 'precio')->textInput(['maxlength' => true]) :"" ?>

    <?= $model->estado == 'vendido' ?$form->field($model, 'comprador')->textInput(['maxlength' => true]) :""?>

    <?= $model->estado == 'vendido' ?$form->field($model, 'fecha_vendido')->widget(DatePicker::classname(), [
    'options' => ['placeholder' => 'Ingrese Fecha ...'],
    'pluginOptions' => [
```

Ilustración 76 Código de formulario para creación y actualización

General

```php
<?php

use yii\helpers\Html;
use yii\grid\GridView;



$this->title = 'Objetos';
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="objetos-index">

    <h1><?= Html::encode($this->title) ?></h1>
    <?php // echo $this->render('_search', ['model' => $searchModel]); ?>



    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
            ['class' => 'yii\grid\SerialColumn'],

            'nombre',
            'descripcion',
            'lugar',
            'fecha_ingreso',

            ['class' => 'yii\grid\ActionColumn',
            'template' => '{view} {contactar}'],
        ],
    ]); ?>
</div>
```

Ilustración 77 Código de la vista general de objetos

Index

```php
$this->title = 'Objetos '.$titulo;
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="objetos-index">

    <h1><?= Html::encode($this->title) ?></h1>
    <?php // echo $this->render('_search', ['model' => $searchModel]); ?>

    <?= GridView::widget([
        'dataProvider' => $dataProvider,
        'filterModel' => $searchModel,
        'columns' => [
            ['class' => 'yii\grid\SerialColumn'],

            'id',
            'nombre',
            'descripcion',
            'lugar',
            'fecha_encontrado',


            ['class' => 'yii\grid\ActionColumn',
            'template' => '{view} {update} {delete}'],
        ],
    ]); ?>
</div>
```

Ilustración 78 Código de la vista del index de objetos

View

```
<p>
    <?php if(!Yii::$app->user->isGuest && $model->us_id == Yii::$app->user->identity->id) { ?>

    <?= Html::a('Update', ['update', 'id' => $model->id], ['class' => 'btn btn-primary']) ?>
    <?= Html::a('Delete', ['delete', 'id' => $model->id], [
        'class' => 'btn btn-danger',
        'data' => [
            'confirm' => 'Estás seguro de querer borrar este objeto?',
            'method' => 'post',
        ],
    ]) ?>
    <?= $model->estado=="perdido" ? Html::a('Devolver', ['actualizar', 'id' => $model->id, 'estado' => 'perdido'], ['
        class' => 'btn btn-warning']) : "" ?>
    <?= $model->estado=="perdido" ? Html::a('Poner En Venta', ['actualizar', 'id' => $model->id, 'estado' => 'venta']
        , ['class' => 'btn btn-warning']) : "" ?>
    <?= $model->estado=="venta" ? Html::a('Vender', ['actualizar', 'id' => $model->id, 'estado' => 'vendido'], ['
        class' => 'btn btn-warning']) : "" ?>

    <?php } ?>


</p>

<?php if($model->estado == "perdido") { ?>

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'nombre',
        'descripcion',
        'lugar',
        'fecha_ingreso',
        [
            'attribute'=>'photo',
            'value'=>Yii::$app->homeUrl.'uploads/'.$model->foto,
            'format' => ['image',['width'=>'100','height'=>'100']],
        ],

    ],
]) ?>
```

Ilustración 79 Código de la vista view de objetos

Implementación de las notificaciones

Configuración del framework

```
'mailer' => [
    'class' => 'yii\swiftmailer\Mailer',
    // send all mails to a file by default. You have to set
    // 'useFileTransport' to false and configure a transport
    // for the mailer to send real emails.
    'useFileTransport' => false,
    'transport' => [
        'class' => 'Swift_SmtpTransport',
        'host' => 'smtp.gmail.com',
        'username' => 'objetosperdidos18@gmail.com',
        'password' => 'objetosperdidos2018',
        'port' => '587',
        'encryption' => 'tls',
        'encryption' => 'tls',
            'streamOptions' => [
            'ssl' => [
                'allow_self_signed' => true,
                'verify_peer' => false,
                'verify_peer_name' => false,
            ],
        ],

    ],
],
```

Ilustración 80 Código de configuración de correos electrónicos

Codificación de las notificaciones

Modelo

Contact

```
public function contact($email, $perdido)
{

    if ($this->validate()) {

            $imagen = $perdido->foto;
            $imagen = '@app/web/uploads/'.$imagen;
            Yii::$app->mailer->compose('@app/mail/layouts/contactar',[
                'content' => $this->subject,
                'content2' => $this->body,
                'content3' => $this->email,
                'imageFileName' => Yii::getAlias($imagen),
            ])
            ->setTo($email)
            ->setFrom([$this->email => $this->name])
            ->setSubject($this->subject )
            ->send();

        return true;
    }
    return false;
}
```

Ilustración 81 Código del método para contactar por email

Controlador

Contactar

```php
public function actionContactar($id)
{
    $model = $this->findModel($id);
    $contact = new ContactForm();

    $email = $this->getEmail($model);

    if ($contact->load(Yii::$app->request->post()) && $contact->contact($email, $model)) {
        Yii::$app->session->setFlash('contactFormSubmitted');

        return $this->refresh();
    }

    return $this->render('contactar', [
        'contact' => $contact,
        'model' => $model,
    ]);
}
```

Ilustración 82 Código del controlador de correos electrónicos

Vista

Contactar

```php
<?php
use yii\helpers\Html;

/* @var $this \yii\web\View view component instance */
/* @var $message \yii\mail\MessageInterface the message being composed */
/* @var $content string main view render result */
?>
<?php $this->beginPage() ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=<?= Yii::$app->charset ?>" />
    <title><?= Html::encode($this->title) ?></title>
    <?php $this->head() ?>
</head>
<body>
                <h1>Sistema de Objetos Perdidos</h1>
                <?php $this->beginBody() ?>
                <h2>Recibiste un mensaje de <?= $content3 ?></h2><br>
                <h2><?= $content ?></h2><br>
                <h3><?= $content2 ?></h3><br>
                <img src="<?= $message->embed($imageFileName); ?>">
                <?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
```

Ilustración 83 Código de la vista para envió de correos

Implementación de las seguridades de acceso

```
        'access' => [
            'class' => AccessControl::className(),
            'ruleConfig' => [
                'class' => AccessRule::className(),
            ],
            'rules' => [
                [
                    'actions' => ['general','contactar','view','create','update','actualizar'],
                    'allow' => true,
                    'roles' => ['admin','user'],
                ],

                [
                    'actions' => ['general','contactar','view'],
                    'allow' => true,
                ],

            ],
        ],

    ];
}
```

Ilustración 84 Código de reglas de acceso